Mastering system and power measures for servers in datacenter

Georges da Costa, Jean-Marc Pierson, Leandro Fontoura Cupertino

Mastering system and power measures for servers in datacenter

Georges Da Costa*, Jean-Marc Pierson, Leandro Fontoura-Cupertino

IRIT, University of Toulouse, France

A B S T R A C T

Using power meters and performance counters to get insight on system's behavior in terms of power consumption is common nowadays. The values coming from these external or internal meters are usually used directly by the research community, for instance to derive higher-level power models with learning techniques or to use them in decision tools such as schedulers in HPC and Cloud Computing. While it is reasonable when one wants only to have a broad view on the power consumption, they can not be used directly in most cases: We prove in this article that the problems of distributed measure and hardware limits are way more complex and create bias, and we give the keys to understand and chose the proper methodology to handle these bias to obtain relevant values for enhanced usage. A generic methodology is analyzed and its main lessons extracted for a direct usage by the research community to master system and power measures for servers in datacenter.

Keywords:
Power measure
System measure
Methodology
Datacenter
Performance counters

1. Introduction

While it is well known that measurements setups, jitter in acquisition monitoring, lost and repeated observations, inaccuracies of different meters, system counters availabilities, processor performance counters monitoring are all bias when dealing with power and system measures, there does not exist a consolidated analysis of these problems. The quality of performance and power measurements relies not only on the available physical infrastructure's accuracy but also on how the experiments are conducted. Interestingly, while everyone would agree on the importance of the process of measurements and the knowledge of the expected accuracy, many do not discuss enough of its importance. In particular, depending on the use cases some inaccuracies can be acceptable or not. For instance, power and performance counters (PMC) measurements' high accuracy is of great importance when used for regression models for creating power models, since the precision of the model will depend on the accuracy of the learning data [15]. Alternatively, when the question is only to estimate the maximum power consumption of an infrastructure of several hundreds of servers, an accuracy of 100 W or 10 s will not even make a difference. In the same vein, a cloud management system having to place, consolidate, migrate services among servers will probably only need accurate enough values every hour and not at high frequency.

For each of the analyzed bias, we will demonstrate through real experiments its impact in terms of accuracy of the power estimates, and the extra power needed when taking it into account. These results serve the community in order to take wise decisions on questions like: which biases should I take into account in a particular case? How to improve the behavior of my platform? We will exhibit lessons learnt for challenges that are commonly faced.

Two aspects will draw particular attention: First, we define a novel model of Power Supply Unit (PSU) power conversion losses. Second, we exhibit the overhead of tracking performance counters on Intel i7 processors and the need to limit the number of concurrent monitoring.

The remainder of this article is organized as follows: Section 2 describes the capabilities of hardware devices used to measure power consumption. In Section 3, the data acquisition infrastructure is detailed. It is followed by Section 4 concerning the power measures bias. Section 5 outlines the problematic of measuring system values. Finally we conclude this research work in Section 6.

2. Power monitoring devices

Hardware power meters are the most accurate source of system's power measurements. The granularity of measurement is crucial for both, power measuring and modeling, and depends on the type of power meter used: external or internal. External meters are placed between the electric outlet and system's power supply unit, while internal meters are located inside the system [24]. Several studies rely on the precision of the power and system

* Corresponding author.
E-mail address: dacosta@irit.fr (G. Da Costa).

http://dx.doi.org/10.1016/j.suscom.2017.05.003 22

monitoring infrastructure to correlate both but without details on the exact methodology. In [33] or in [20] for example, authors model power consumption of VMs without explaining in details the bias of the measuring infrastructure and its impact on the resulting models.

2.1. Intra-node devices

Fine-grained measurements can be achieved by embedding power sensors into the system, enabling device specific measurements. Such technique monitors device's direct current (DC) power rails to decouple its consumption from the system's power. The measurements can be done using shunt resistors or clamp meters. Shunt resistors are placed in line with each power rail to measure the voltage drop across the resistor, allowing current and power calculation [25]. The resistors need to be carefully chosen to give high precision and low power loss; besides, if the voltage provided by the rail varies, additional components are needed to measure it as well. Likewise, clamp meters are placed around each power rail without disconnecting the wire, providing a less intrusive power measurement [3].

Despite the use of such techniques during product design and testing phases, manufacturers rarely incorporate internal meters into commodity products, due to additional costs and increased board area. PowerMon2 [2] is a stand-alone monitoring device placed between system's PSU and its devices. It fits in a standard 3.5″ hard drive bay, monitoring voltage and current on DC rails in a sampling rate of 1 kHz through a USB interface. Similar approaches can be found in PowerPack [16] and SEFLab [13]. PowerPack is a framework to isolate the power consumption of devices including disks, memory, NICs, and processors in a high-performance cluster and correlate these measurements to application functions. The measurements are done through the use of a shunt resistor for each power rail, detecting the power of each device. The information of each device's power consumption is used to evaluate the impact of DVFS techniques on clusters.

Even though, some vendors provide integrated monitoring solutions on their hardware. Intel RAPL (running average power limit) provides power measurements for recent processors. While the power was previously modelled in the Intel Sandy Bridge family, real measurements are now given by RAPL in the Haswell chips, using this measurements for hard power capping. In [17] Hackenberg et al. study the energy efficiency features of Haswell chips. One of the major accelerators used in HPC applications nowadays, GPUs explore their high performance per watt capability on their marketing campaigns. Nvidia's recent graphic cards contain embedded power sensors, providing power usage of the entire board (GPU and memory). The data can be retrieved in milliwatts with a 5% accuracy through the NVIDIA Management Library [27]. Another common practice is to include such sensors in blade servers. Dell's PowerEdge M1000e [7] enables real-time reporting for enclosure and blade power consumption, providing a total chassis power inventory including power supplies, iKVM, I/O Modules, fans and server modules. RECS (Resource Efficience Computer System) computebox [29] also includes embedded meters for each of its modules with 1 W accuracy.

2.2. External devices

The most architecture independent and less intrusive method is to measure alternative current (AC) power at the outlet. External power meters measure the power consumed by the entire system. Small-scale environment can be deployed with general purpose solutions such as Plogg [11], Kill A Watt [28] and watts up? [10]. The data measurements can be retrieved through serial, Ethernet or even BlueTooth connections depending on the model. In [14], the authors introduce PowerScope, a methodology to collect detailed energy use per process using external meter and customized system calls.

For large-scale deployment, such as in data centers, intelligent PDUs can be installed in rack cabinets [1,8]. These are standard rack PDUs with embedded power meters for each power socket, which can be monitored through a serial or an Ethernet port. Although the requirement of additional investment, intelligent PDUs are easy to deploy and provide a valuable information regarding AC power consumption in a data center, which can be used by managers to improve data center's energy efficiency. In addition, AC power is used by energy providers to charge their clients. Therefore, if the economical aspect of energy efficiency in a data center will be evaluated, AC power needs to be estimated. Different methods are available to obtain the measured values depending on the goal. IPMI (Intelligent Platform Management Interface) as an example is often used for detecting anomalies and cannot be used for precise measurement.

However, external power meters measure power in a coarse-grain fashion, i.e. only the system level power is monitored. They cannot decouple the actual used power from the power wasted due to inefficiencies of the PSU, i.e. during AC to DC conversion. When using such meter as target for models' creation, this lack of information regarding conversion losses adds noise to the measurements, impacting the quality of power models.

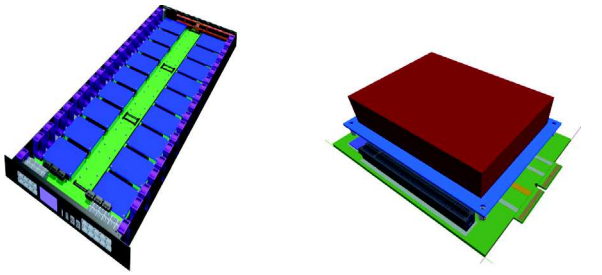
3. Data acquisition infrastructure

Power and performance monitoring increases system's processing overhead, either to gather target system's Key Performance Indicators (KPIs), or to connect to a power meter. To avoid adding more noise to a complex system during the data gathering, an independent monitoring node, i.e. a data acquisition server (DAQ server) is required. This monitoring server is responsible for fetching power measurements from the remote power meter during the execution of a given load, and synchronizes it with the performance indicators collected on the target architecture. Accurate measuring requires that the internal clock of all servers are synchronized before the monitoring starts, this is achieved using a Network Time Protocol server. Moreover, other synchronization issues exist and will be tackled later in this chapter.

In order to evaluate the measure precision and related impact, we used as a demonstration testbed a high-density server with internal and external power meter at different levels. The available testbed consists of a high-density server[1]; a Power Supply Unity (PSU); an external power meter (Plogg, a bluetooth powermeter); and an independent monitoring node (DAQ server). The RECS server monitors internally KPIs (performance values such as processor load, memory or any other relevant value) but also uses an internal power meter in-between the PSU and the motherboard.

The RECS server is composed of 18 modules connected through a back-plane controller in a single 1U rack chassis. Each of its modules operates as an independent computing node, connected to a central back-plane through a COM Express based main-board. This enhances the server's reconfiguration capabilities, allowing the use of any available COM Express main-board with the basic size, to be plugged in as a module. Moreover, embedded in each module there is a thermal and a current sensor to measure its temperature and power, respectively. The central back-plane forwards the network's traffic of each module to the front panel of the server though a Gigabit Ethernet Network. In addition, it also connects the modules' micro-controllers with the central master micro-controller.

[1] RECS from Christmann www.christmann.info.



(a) RECS server (b) RECS module

Fig. 1. Geometry of the RECS high density computing system [30]. The RECS server (a) is composed of 18 independent modules (b) arranged in two rows.

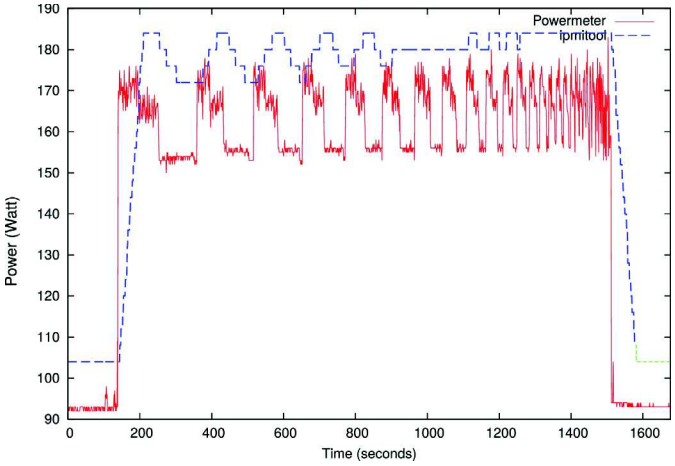


Fig. 2. Comparison between power measured using IPMI and a precise wattmeter for a changing workload.

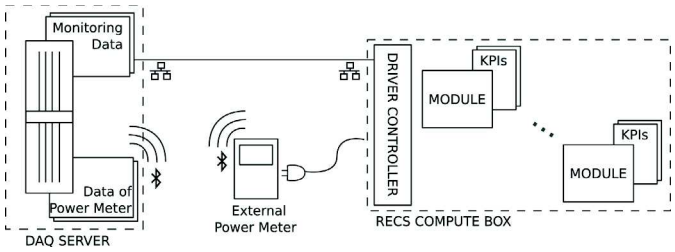


Fig. 3. Data acquisition's infrastructure. The monitoring server (DAQ server) uses different communication techniques to gather data from the remote power meter and the server, generating disparate response times.

Module's micro-controller switches the node on/off and reads its power and temperature measurements through dedicated sensors. The geometry of a RECS module and a RECS server is shown in Fig. 1.

A schematic view of the data acquisition infrastructure is presented in Fig. 3. The DAQ server communicates with the Plogg and the server through Bluetooth and Ethernet connection, respectively; while the KPIs are logged locally in the modules. All file logs are synchronized after the benchmark execution, to not interfere neither on its network, nor its CPU overhead. This method ensures a low memory impact in order to have a negligible impact on the running applications.

Even if the rack in which the high-density server was equipped with an IPMI monitoring tool, it has to be noted that such values cannot be considered as precise measurement of the power consumption. As shown in Fig. 2, IPMI usually returns a higher value with some latency. Indeed, IPMI has been designed in order to detect problems and to raise alarms. IPMI usually provides overestimated values changing infrequently.

Table 1
Performance indicators (KPI) and External Power measurements' response times.

	Min.	*Mean*	σ	Max.	Samples
External power	0.3189	0.6162	0.1894	1.646	3810
KPI	0.0021	0.0060	0.0047	0.052	2537

In our site, we have a hybrid server with six i7 and twelve Atom modules. Each i7 module contains an Intel Core i7-3615QE processor with 16 GB of RAM and an Intel 82579LM Gigabit Ethernet, while each Atom module has an Intel Atom N2600 processor with 2 GB of RAM and a Realtek RTL8111 / 8168B PCI Express Gigabit Ethernet controller. All nodes are disk-less and boot the same OS image – Scientific Linux release 6.4 with kernel v2.6.32. Scientific Linux is a Red Hat Enterprise Linux rebuild sponsored by Fermi National Accelerator Laboratory [12]. Each RECS module contains an integrated fan with only two operational modes: on and off, i.e. it does not feature adaptive cooling using fan speed control. The fan is switched on/off by the node's micro-controller at the same time as its motherboard; when turned on, it consumes approximately 6 W of DC power.

Challenge I: How to synchronize different timeseries?

The data acquisition process is creating three log files: Internal Power, External Power and KPI. The Internal and External Power logs are built in the DAQ server, while the KPI is created in the target platform. Each log file contains its measurements, along with request and response timestamps. The data acquisition method includes some post-processing procedures to enhance the accuracy of the measurements (detailed in Section 4).

The synchronization between power and KPIs measurements needs to consider the external meter's limitations, i.e. communication latency, time delay and sampling rate. In the case of our infrastructure (the Plogg), the communication latency varies accordingly to Table 1. This experiment (a set of micro-benchmarks stressing more and more the CPU, memory, network, used to challenge the platform) lasts 2537 s, the KPI are collected at 1 Hz frequency. One can see that the time required to fetch KPI's measurements are almost 100 times faster than the External Powermeter's data, i.e. 6 against 616 ms, and with a higher variance. Internal power measurements are not shown in the table as usually it is not possible to have access to the internal code of the internal wattmeter and thus, there is no way to measure its latency.

To provide accurate values for the high latency of the Plogg meter, the power measurements are done at its higher frequency, creating a log file with time-stamps of the requested (t_{req}) and retrieved (t_{ret}) times for each measurement. However, KPIs' measurements are quite fast to be fetched and can be considered instantaneous. Thus, they were gathered at the lowest rate possible (1 Hz) to avoid system's overhead. Therefore, during any experiment, one can also note from Table 1 that the number of samples for both are different: Indeed the Plogg is not providing data at 1 Hz. If one wants to get a correct snapchat of the running platform, so that to identify the relationships and correlations between power and PKIs data, it is mandatory to synchronize them. Due to the Plogg's low latency, the measurements for KPIs and power are only synchronized once per second. After the workload execution, all logged data files are synchronized based on their closest timestamps, as follows:

$$\forall i, j \in [0, t_d]$$
$$arg\min_{KPI_i, Plogg_j} \frac{(t_{req}^{KPI_i} + t_{ret}^{KPI_i})}{2} - \frac{(t_{req}^{Plogg_j} + t_{ret}^{Plogg_j})}{2} \quad (1)$$

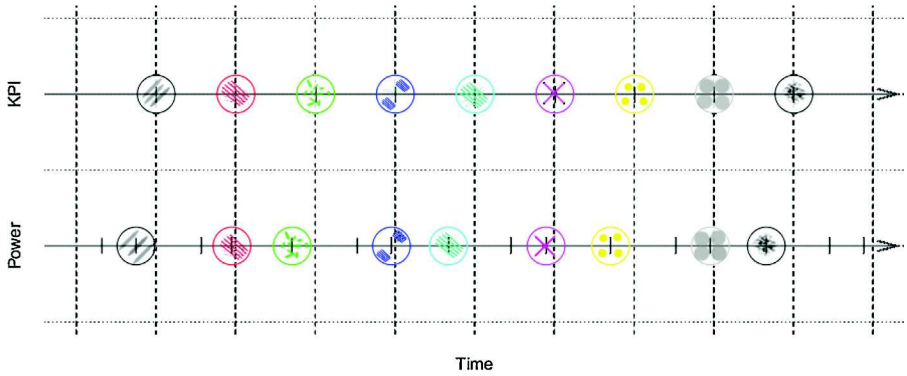


Fig. 4. Power (from Plogg) and performance (KPI) data synchronization. Circles having the same color and pattern represent the synchronized data.

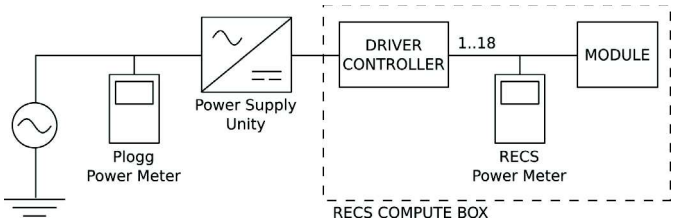


Fig. 5. Power metering infrastructure using a Plogg and 18 RECS embedded meters.

where t_d is the time duration and KPI_i and $Plogg_j$ are the KPIs and power measurement.

Fig. 4 represents a real synchronization case, where lines represent averaged response times and the circles having the same color are synchronized together. One can see that some of the Plogg's measurements will be dismissed.

Lesson I: Synchronisation by closest timestamps using Eq. (1)

4. Power measuring

Power measurements' accuracy is of great importance when using the power as explanatory variable of regression models, since the accuracy of the model will depend on the quality of the learning data. To provide precise measurements, we implemented a power measuring infrastructure that extends the capabilities of the RECS server by adding an external power meter to measure the power drained at the outlet level. The power metering infrastructure schema is presented in Fig. 5, which depicts where the Plogg and RECS embedded power meters are placed to acquire outlet- and module-level measurements, respectively.

Our server comes with one power sensor per module, summing up 18 sensors. Each sensor can measure the power dissipated by a module with a 1 W precision. RECS server's micro-controller-based monitoring architecture is accessible to the user through Ethernet connection by a dedicated network port. The data acquisition requires a single request to gather information about all installed modules. Although efficient to provide high throughput, the monitored values are updated once per second.

Plogg is a low cost power meter with a power outlet which makes it easy to deploy any device fed through a power plug. Its small size and deployment simplicity makes it a device that can be used to measure different computer system's power in a heterogeneous data center. In [22], the authors compare several low cost power meters, reporting that Plogg was the most accurate device with 1.5% average error. In fact, later experiments shows that the Plogg power meter provides more accurate measurements than the RECS embedded meters. Plogg uses Bluetooth communication to transmit consumption information, but the time monitoring frequency is kept the same, providing the same average value per second.

Table 2
Power meters' communication latencies.

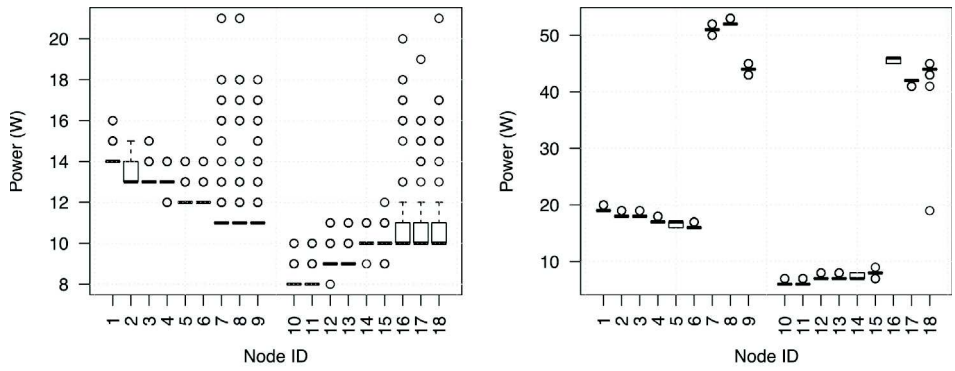
Power meter	Min.	Mean	σ	Max.	Samples
Plogg	0.3199	0.6164	0.1844	1.637	1513
RECS	0.0529	0.0550	0.0055	0.074	1009

Due to the different communication protocols, the time to fetch data from the Plogg and RECS meters varies significantly. An experiment profiling their response times was done measuring the elapsed time between each meter's request and response. Table 2 shows the latency for accessing each power meters' measurements. Due to the Bluetooth connection, Plogg presents an average latency of 0.6 and can reach up to 1.6 s to fetch a single value. The RECS embedded meter provides an Ethernet connection which provides stable response time of 0.05 s.

Challenge II: How to choose between internal and external power-meter

RECS embedded power meter is noisy, presenting some measurement errors. Fig. 6 shows the box-plot of 1000 power measurements using the RECS meter, when idle and during the execution of Linux's `stress` command in all cores of all nodes. As all nodes run the same load, the power dissipation per module type is expected to be the same. However, the results show a high variation of the values for both Atom (IDs 1–6 and 10–15) and i7 (IDs 7–9 and 16–18) modules. When the systems are idle, Fig. 6(a), the measurements are quite noisy, presenting several outliers in the box-plot, up to 11 W. With the system under stress, Fig. 6(b), the variance of the measurements for each module is more visible (with almost no outliers), reaching almost 10 W difference between the two box-plots of two similar modules (see nodes 8 and 17 for instance).

Although acceptable for data center level measurements, the low precision of the RECS embedded meter would have a huge impact on a power model creation. The vendor claims that the embedded watt meters have a 1 W precision per module, but our experiments show that this error can be much higher. To compare the accuracy of both meters (RECS and Plogg), a CPU intensive experiment was conducted. Only one i7 node of the RECS server was turned on. All cores were set to operate at their maximal available frequency, while the processor load was increased in 2.5% (10% per core) step using Linux `stress` command. The values reported from each RECS module were aggregated and plotted against the values from the outlet. The results in Fig. 7 points out the impact of meters' accuracy on the measurements. The solid lines represent the measured data. The big gap between RECS and Plogg measurement lines comes from power conversion losses and the back-plane power consumption, which are not taken into account when using



(a) All nodes idle (b) All nodes running `stress`

Fig. 6. RECS embedded power meter line issues. IDs 1–6 and 10–15 are Atom modules, IDs 7–9 and 16–18 are I7 modules.

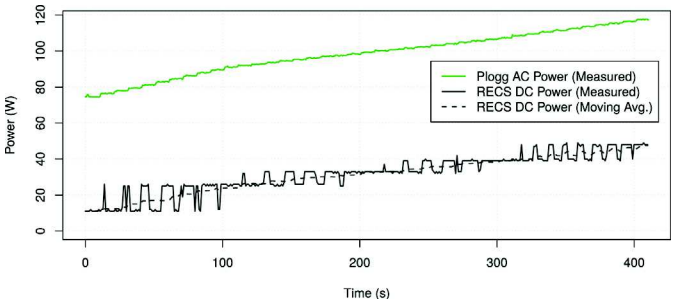


Fig. 7. Comparison of embedded (RECS DC power) and external (Plogg AC power) watt meters for a workload which sequentially increases the processor's load.

the embedded RECS meters. One can see that while the Plogg measurements increases in a step forward manner while the embedded meter presents a lot of oscillation, creating a lot of noise to be used as a target for a model creation. The noisy data from RECS meters incurs in discontinuities, making it difficult to extract knowledge from data. RECS measurements were then filtered using a moving average (dashed line). The moving average avoids sharp variations on the measurements depending on the sliding window. The value of the sliding window (15) was carefully chosen by measuring the distance between peaks. The results of the moving average show a high correlation with the measurements from the Plogg meter. However, moving average cannot be used during model creation since it avoids sharp variations that may actually exist when measuring the power consumed by hardware.

Lesson II: Internal powermeters are limited to coarse grained measures

4.1. PSU's power conversion losses

Challenge III: How to take into account PSU efficiently?

Power supply units (PSU) always waste power during current and voltage transformations. As far as we know, PSU's conversion losses have never been considered while proposing new power models. In the literature, authors usually choose between internal or external meters, without handling AC to DC conversion losses when exploiting external powermeter measurements. Even though the reported errors of fitted model are small, there is no in-depth knowledge of where does this error comes from.

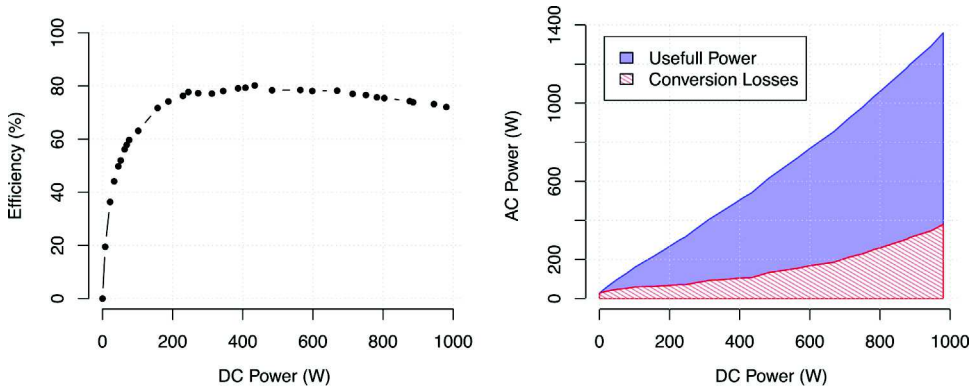
A simple way to model power losses is to use the average efficiency rate, like those proposed by the 80 Plus label, and to create a simple linear model. However, PSU's efficiency is not constant over its entire input range, which implies that its AC to DC conversion losses need to be more deeply modeled. In this section we propose an uni-variate polynomial modeling of PSUs to eliminate the conversion losses from the power estimation errors when dealing with external power meters. The order of the polynomial is defined based on experimental data as will be detailed later in this section.

The PSU used to supply the needed 12 V DC for the RECS server is based on six single Power Units. To model the electrical characteristics, a set of measurements has been done by the manufacturer [30]. These measurements compare the PSU's input (P_{AC}) and output power (P_{DC}), determining the load dependent efficiency as the ratio P_{DC}/P_{AC}. Although, in this case, the data used to model the PSU losses was provided from its vendor, the use of a clamp meter will provide a non-intrusive solution to measure AC/DC conversion losses when the data is not available. The difference between the input and output power is due to the power dissipation which is converted to heat emitted to the air. The results in Fig. 8 present PSU's efficiency and power losses based on server's DC power request. One can see in Fig. 8(a) that the efficiency is nonlinear, presenting low efficiency for small power loads, a barely constant efficiency for a large range of load and a decrease in the upper range. The low efficiency of the lower load may be explained by the current leaked in the electronic circuitry, which is the same for any range, but has a bigger impact for small loads. At the other edge, high loads will overheat the PSU due to a limit of the fans speed, increasing the leakage power, once again. The most efficient operating range of the PSU is between 200 and 800 W DC, achieving around 80% of efficiency. As the RECS server used in our experiments operates from 60 W AC (20 W DC) (all modules turned off) to 600 W AC (450 W DC) (all nodes fully stressed using a CPU intensive workload), the measurements when using a single node will be in the transient phase (below 200 W DC). Fig. 8(b) presents the amount of power lost during the current conversion. The power losses can vary from 26 to 380 W per server, which will have a significant impact when using AC power data to generate power models.

We propose a solution where the actual power usage of the RECS server can be estimated using AC power measurements by modeling the PSU's power losses. To handle such losses, we use a uni-variate polynomial model. The order of the polynomial was defined by creating a linear regression of the power at the input and output of the PSU while sequentially incrementing its order until the regression's residuals start to increase (following method described in [6]). Once the degree of the polynomial is defined, variables having the highest p-values were sequentially excluded from it until the residuals start to increase once again. The achieved polynomial is given as follows:

$$P_{DC} = w_0 * P_{AC} + w_1 * P_{AC}^2 + w_2 * P_{AC}^3 + \ldots + w_7 * P_{AC}^8 \quad (2)$$

where $\mathbf{w}$ is the vector of weights computed by the linear regression, P_{AC} and P_{DC} represents the AC and DC power, respectively.



(a) Efficiency (b) Conversion Losses

Fig. 8. RECS 2.0 PSU's power conversion profile.

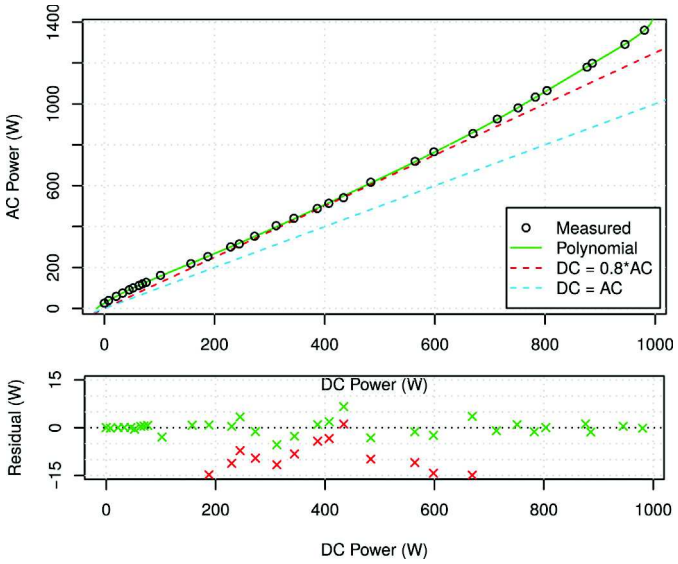


Fig. 9. PSU model's data fit and residuals. The reference line ($DC = AC$) corresponds to the estimations when no modeling is done.

Fig. 9 shows the PSU estimated and measured power along with their residuals for the proposed polynomial model, a simple 80% efficiency based model ($P_{DC} = 0.8P_{AC}$) and a reference model ($P_{DC} = P_{AC}$). One can notice that the residuals for the reference model does not even appear in the residual graph since it varies from −26.4 to −379.7 W, evidencing one more time the need to model PSU's losses. The 80% efficiency model provides more realistic approximation providing errors smaller than 15 W in the (187, 670) DC range, but achieve errors up to 107.6 W. The polynomial model presents a correlation of 1, i.e. an almost perfect fit. Its residuals vary from -5.3 to 6.6 W with a standard error of 2.46, these residuals can be neglected if compared to the other model's residuals. The residuals of the polynomial model reach at most 2.88 % of error. Thus, the polynomial model is considered to have good predictive ability.

The evaluation of the impart of modeling PSU's losses for a single node is shown in Fig. 10 where the presented model is the DC power (estimated) blue line. The lines on the graph represent the dynamic power, while running the workload, i.e. the actual power minus the idle one. One can see that, removing the power losses from Plogg's DC Power, the measurements get closer to the embedded meter measurements from RECS's DC Power. This enhances the importance of decoupling the power losses from the data used to create power models.

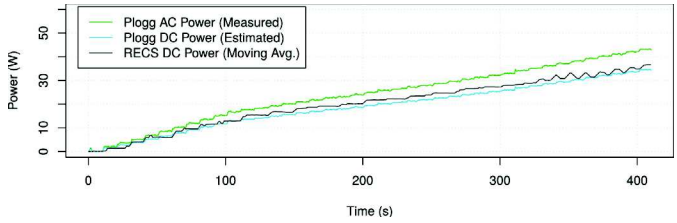


Fig. 10. Comparison between measured and estimated powers. The plotted values correspond to the dynamic power, i.e. actual minus idle power, for the same workload as Fig. 7.

Lesson III: Polynomial model (Eq. (2)) is efficient for PSU modeling

This methodology can be used to model any PSU conversion losses, enhancing the accuracy of power models independently of the chosen power modeling method. The PSU modeling can be done either based on the vendor information, or using an external meter to measure the input power and a clamp meter to measure its output power under different power loads, providing the input data to create the PSU's losses model.

Interestingly, some home appliances already come with multiple circuits in PSU which are switched on and off according to its usage, e.g. idle and active power of televisions, decreasing the impact of leakage power for small loads. Similar approaches may be implemented for computing systems, requiring the composition of several formulas to proper model the switch.

Alternatively, besides following the proposed method, RAILS (Redundant Array for Inexpensive Load Sharing) from PowerNap's authors [26] proposed to use several smaller PSU instead of one big, where each smaller one is efficient in a smaller zone, but their assembly allows for larget power distribution. In this way, losses are decreased at the cost of using more PSU.

4.2. Timing jitter

Challenge IV: Keeping synchronicity of hardware with independent clocks on large-scale experiments

When monitoring the power, its measurements may be delayed in time due to the meter's limitation. In electronics and telecommunication, this deviation from the true periodicity of a expected periodic signal is called jitter. The jitter can be random or deterministic. The Plogg device presents a deterministic jitter, i.e. the time latency is kept the same for an entire experiment. To determine whether the power measurements are delayed on time, a load pattern is included at the beginning of each workload monitoring. This pattern stresses the processor in a constant time interval

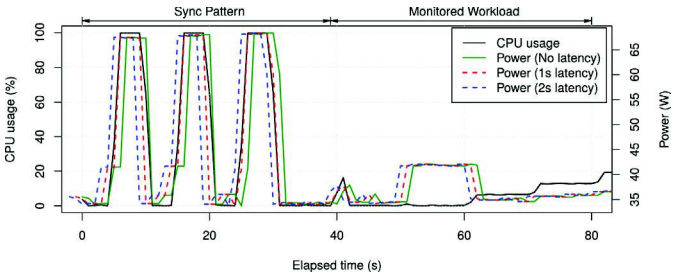


Fig. 11. The impact of time latency on the data. Time jitter can be removed using a synchronization pattern before the execution of each workload.

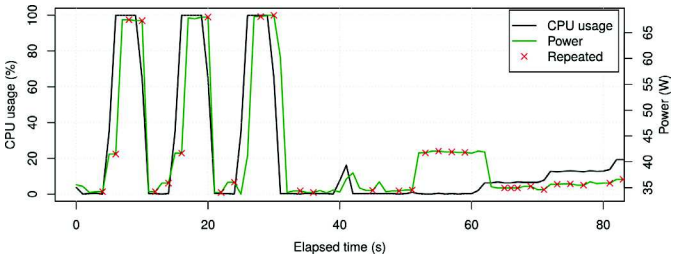


Fig. 12. The Plogg's power meter provides some repeated values.

and the data gathered during its execution is used to determine the time latency of the power measurements and to synchronize them with the KPIs. The importance of time synchronization is presented in Fig. 11 by plotting the processor usage along with the power consumption shifted in time. The first 40 s are due to the synchronization pattern that is added in all workload and allows us to determine the time latency between KPIs and power. Similarly, a pattern is added at the end of the execution (not shown on the figure). The synchronization pattern's Mean Average Error (MAE) is 0.16, 0.05 and 0.15, when considering 0, 1 and 2 s latency for the power measurement, respectively. The results show that data is actually delayed and provide a better fitting when considering 1 s latency. After synchronizing, the synchronization pattern data is removed from the workload data.

Lesson IV: Using starting and ending patterns makes readjustment simple

4.3. Repeated values

Challenge V: Finding artifacts

Another issue when dealing with the Plogg meter is that when a request arrives and for some unknown reason the device is not able to measure the power, it provides the last measured value. As the Plogg watt meter have a milliwatt precision, it is easy to identify when this problem happens because even for two similar consecutive measurements, their values are not likely to be identical. The repetition of the data adds noise to the data and may cause some false validations. Thus, all data entry which are repeated power measurements must be removed from the acquired dataset. Fig. 12 shows the repeated values during the initial execution of a workload. One can see that the number of invalid repeated values is quite significant, representing 32% of the total power measurements for this case. It also important to notice that the repeated values do not follow a pattern; thus, they cannot be avoided by simply changing the sampling rate.

Lesson V: Harness the inherent hardware characteristics such as lack of stability

5. Performance measuring

The modules are the main actors of the RECS server, executing the workload and monitoring performance indicators either to create a model or to estimate applications' power consumption. A modular library of sensors and power estimators, called Energy Consumption Library (`libec`) [5,4], was developed to accurately measure KPIs with low overhead.[2] The main goal of `libec` is to aid the development of new power estimators. To make it easy to extend and maintain, it was implemented in C++ and distributed under the GNU General Public License (GPL) version 3.0 or later. This library contains a set of performance indicators, or sensors, which can be used as input variables for several power models. The information provided from the sensors comes mainly from Linux kernel's API and the `/sys` and `/proc` file systems. Nevertheless, these sensors can be extended in order to collect different data coming from any source specific sensors.

Sensors in `libec` can be implemented at two levels: system and/or process. Process-level sensors can be directly associated with a process identifier (PID), having a straight relationship with software usage. Usually, every process-level sensors may be ported to system-level by aggregating all running processes' measurements, the reciprocal is not true. System-level sensors measures not only the aggregated value for all the processes, but also some physical properties that cannot be decoupled and associated to a PID, such as CPU thermal dissipation. In addition, the library provides application-level sensors to estimate application's power consumption, which will be later extended to include the results presented on this research. A complete list of explored performance indicators can be found in Table 3. In this table the KPIs are categorized into OS information (SYS), hardware (HW), software (SW) and cache memory (CM) performance monitoring counters (PMC) and model-specific registers (MSR). The concept and implementation of each available sensor will be further described based on its category.

5.1. Operating system information

The operating system has a strategic position to profile every device performance. Since OS kernel operates devices' drivers, it can accurately monitor their interaction with the system. Different from other performance indicators that provide mainly processor related information, such as PMC and MSR, OS can provide fine-grained events of several components, such as memory, disk and network card. Monitored events are made available in user-space through the `/sys` and `/proc` file systems, some of them may require specific kernel modules or patches. The OS events used during the data acquisition are described as follows:

- The amount of time a system or a process spends inside a processor can be measured as the time it has been scheduled in kernel and user mode times. Time spent in different processor mode can be fetched, in jiffies, from `/proc/stat` and `/proc/[pid]/stat` for system- and process-level, respectively. The size of a jiffy is determined by the value of the kernel constant HZ, which can be retrieved in user-space using the command `sysconf(_SC_CLK_TCK)`. The kernel version used on the experiments presents a constant of 100 Hz, meaning that the time is reported in a granularity of 1 sample per 10 ms. Time is not converted from jiffies to seconds to not lose precision. CPU elapsed time is measured by subtracting the previous from the current

[2] Available for download in http://github.com/cupertino/ectools.

Table 3
Performance indicators divided into several categories: hardware (HW), software (SW) and cache memory (CM) performance monitoring counters (PMC), OS information (SYS) and model-specific registers (MSR).

Type	Name	Name
SYS	cpu-time	cpu-pstate
	ram-usage	cpu-usage
	net-snd-bytes	cpu-cstate
	net-rcv-bytes	
MSR	cpu-pstate-msr	cpu-cstate-msr
	cpu-temp	
PMC-SW	cpu-clock	context-switches
	major-faults	task-clock
	cpu-migrations	alignment-faults
	page-faults	minor-faults
	emulation-faults	
PMC-HW	instructions	cache-misses
	idle-cycles-frontend	branch-instructions
	cpu-cycles	idle-cycles-backend
	branch-misses	bus-cycles
	cache-references	ref-cycles
PMC-CM	L1-dcache-loads	L1-dcache-load-misses
	iTLB-loads	L1-dcache-stores
	L1-dcache-store-misses	iTLB-load-misses
	L1-dcache-prefetches	L1-dcache-prefetch-misses
	branch-loads	L1-icache-loads
	L1-icache-load-misses	branch-load-misses
	L1-icache-prefetches	L1-icache-prefetch-misses
	node-loads	LLC-loads
	LLC-load-misses	node-load-misses
	LLC-stores	LLC-store-misses
	node-stores	LLC-prefetches
	LLC-prefetch-misses	node-store-misses
	dTLB-loads	dTLB-load-misses
	node-prefetches	dTLB-stores
	dTLB-store-misses	node-prefetch-misses
	dTLB-prefetches	dTLB-prefetch-misses

time. This indicator is used to filter the active processes as it will be seen later.

- Processor' usage can be estimated, in terms of load percentage, by the ratio between CPU and total elapsed time. A formal definition is provided in Eq. (3), where t_{sys}, t_{usr} and t_{idl} are the system, user and idle time, respectively. Although cpu-time information regarding each core activity at system-level is available, process-level times available in the `/proc/[pid]/stat` file do not distinguish between cores, i.e. one cannot precisely determine the processor's core usage of a process. Estimations may be done using the last core on which the process ran, but this will not take in account possible context switches that may happen during a time period.

$$CPU_{use} = \frac{t_{sys} + t_{usr}}{t_{sys} + t_{usr} + t_{idl}} \tag{3}$$

- Processor's performance states (P-States) defines its operating frequency through the DVFS (Dynamic Voltage Frequency Scaling) technique. Operating system's requested frequency can be retrieved in KHz from the virtual `/sys/` filesystem.[3] One should notice that this file only provides the requested frequency, not the actual frequency itself. Information regarding Intel's Boost Technology frequencies, for instance, cannot be fetched this way. Besides, although recent operating systems allow different frequencies requests for each processor's cores, many architectures do not feature independent core frequencies, feeding all cores at the same frequency when in active state, in this case, the information from `scaling_cur_freq` will be far from reality. Some hardware allows the kernel to fetch the actual processor's core operating frequency using the `cpuinfo_cur_freq` file, this is not the case in our environment.
- Processors have different idle power-saving states (C-States) which defines the processor's idle units to be shut down. The number of power states may vary according to the target hardware, the consensus is that in all hardware C0 is the active state, i.e. when the processor is fully turned on. Time spent on each CPU power state can be fetched from the virtual `/sys/` filesystem.[4] This file does not have a good precision for C0 when the system is highly loaded.
- Memory usage can be measured in system- and user-level. System-level usage is measured by extracting the free from the total memory provided by the `/proc/meminfo` file, the available field present in this file is not used because it only provides the amount of memory available for userspace allocation without causing swapping. At the process-level, the portion of its memory that is held in RAM is provided by the resident set size variable in the `/proc/[PID]/stat` file.
- Networking received/transmitted packets/bytes flow can be retrieved in system-level from the `/proc/net/dev` file. The user must define if the retrieved data will come from the sum of the networking interfaces or from just one of them. Process-level information require kernel patches or modules such as netatop [21].

5.2. Model-specific registers

Intel introduced model-specific registers (MSRs) in Pentium processors as experimental test registers. Nowadays, most processors from all vendors contain MSRs that overcome the experimental behavior, storing data and setting information for the CPU. MSRs provide control for a number of hardware and software-related features, such as performance monitoring counters, debug extensions, memory type range registers, thermal and power management, instruction-specific support, processor feature/mode support [19]. The MSRs can be read and written in Linux platforms through the `/dev/cpu/[cpuID]/msr` file interface. A given MSR may not be supported across all families and models of processors.

- Processors contain embedded digital temperature sensors which can be read through MSRs. A kernel driver, namely `coretemp`, works on Intel processors by reading the `IA32_THERM_STATUS` MSR [19]. This component dynamically creates a native events table for all the sensors, providing per core input temperature at `/sys/bus/platform/drivers/coretemp/coretemp.[coreID]/temp1_input`.
- Processor's cores' frequencies can also be retrieved by MSRs. The average frequency over a period of time can be precisely calculated, including Boost frequencies, observing a pair of MSRs available in recent X86 processors [19]. MPERF and APERF increase with the maximum and actual frequency in C0, respectively. Each core's operating frequency can be estimated using MPERF and APERF MSRs as follows:

$$f_{coreID} = f_{\max} \times \frac{APERF}{MPERF}. \tag{4}$$

- Time spent in active state (C0) is another variable that can be more accurately measured using MSR. It can be estimated using the same MPERF MSR as the cpu-pstate-msr, as follows:

$$t_{coreID,C0} = \frac{MPERF}{t_{total}} \tag{5}$$

[3] `/sys/devices/system/cpu/cpu[coreID]/cpufreq/scaling_cur_freq`.

[4] `/sys/devices/system/cpu/cpu[coreID]/cpuidle/state[stateID]/time`.

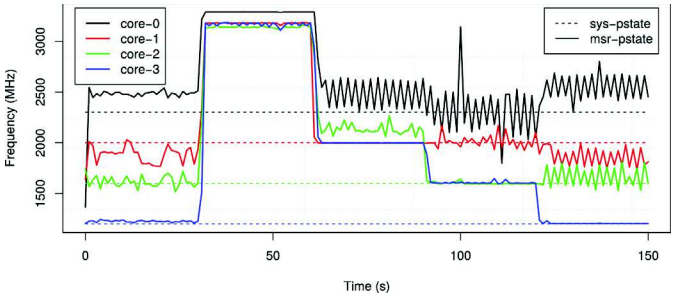


Fig. 13. Frequency measurements using operating system (sys-pstate) and MSR (msr-pstate) information.

Challenge VI: When different sources are available for a sensor, which one to choose?

The accuracy of `cpu-pstate-msr` performance indicator was analyzed while comparing it to the operating system's frequency. An experiment run in an i7 module stressed the system differently in 30 s time steps. Each processor's core was set to operate in a different frequency; more precisely core 0, 1, 2 and 3 were set to Boost, 2.0, 1.6 and 1.2 GHz, respectively. At first the system was kept idle, and then one core was fully loaded using Linux's `stress` benchmark for 30 s each (e.g. for 30 s core 0 was stressed, then core 1 was stressed for 30 s, etc). As the execution of a process can be switched between cores, processor's core affinity of the `stress` process was changed sequentially from core 0 to 3 through the `taskset` command. Processor's core affinity is a scheduler property that associates a process to a given set of cores on the system. The results of the experiment are sumarized in Fig. 13.

During the first 30 s, all cores operate near to the specified frequency when the system is idle, except for Boost mode. Once a core is under stress the processor sets cores' frequency depending on the overloaded core's setup. When the stressed core is in Boost mode, its frequency raises to 3.3 GHz, which is coherent with Intel's data-sheet [19]. When stressing all subsequent cores, the stressed core has its frequency set precisely to the requested frequency (sys-pstate).

Interestingly, it is important to notice that other cores may also change their frequencies, even when not stressed and not asked for: When a core is overloaded, all cores that should operate in higher frequencies do not change their frequencies (normal behaviour), although those that should operate in lower frequencies gets the same frequency as the stressed core (and then gets higher frequencies than expected, leading finally to additionnal power dissipation).

Even if MSR's values present a significant noise when the core is idle, operating system ones do not properly measures core's frequency when in Boost mode and consider that the cores operates independently, which depends on the number of core's voltage regulators available [18], which is not always true. It is important to carefully understand the behaviour of the sockets and cores at hand when conducting experiments.

Lesson VI: Hardware level sensors (e.g. MSR) are usually more precise than system level ones

5.3. Performance monitoring counters

Low level event counters are often used to detect software bottlenecks and improve their performance. As stated in Section 5.2, performance monitoring counters are a special kind of MSRs. These counters have no immediate relation to energy consumption, being a good metric to be used as power model's input. As some MSRs may not be present or do not have the same address for different processors' architecture, the Linux kernel includes some standard counters presented in several architectures. The `perf_events` kernel API provides a simple interface to access such counters and has been part of the kernel since version 2.6.31 [9]. It classifies events into software or hardware counters. A detailed description of each PMC can be found in the Linux man pages [23]. Table 3 list the PMCs used during this research. Moreover, recent Linux kernel versions, such as 3.15.6, provide tracepoint event counters for KVM and Xen virtualization engines, enabling the use of our methodology within Cloud infrastructures as well.

Software event counters, listed as PMC-SW in Table 3, are OS level counters and only require kernel implementation of such counters. There is no restriction on the number of concurrently observed software events; the same does not apply to hardware events.

Processors have embedded performance monitoring units (PMU) to provide hardware event counters, see PMC-HW and PMC-CM in Table 3. Some PMUs are generic and available in most architecture, although some of them can be non-deterministic, most of the measurements present a small standard deviation [32]. The quantity of PMUs present in hardware will impact its size, cost and power consumption; thus, depending on the target audience, the processors' manufacturer decides whether to include such PMUs or not.

The number of hardware counters present in the processor is limited. Hence, to allow a larger number of monitored counters, the kernel will automatically multiplex them over time whenever the number of concurrently monitoring events surpass the actual number of counters [31]. Time-division multiplexing only applies to PMU events, providing for each event a chance to access the monitoring hardware. With multiplexing, an event is not measured all the time, underestimating its values. Generic PMUs may be fixed, programmable or not available depending on the processor. Table 4 lists hardware performance events available in the target machine nodes. Accordingly to the `perf` application, the Atom and i7 processors used in our test-bed support a maximal number of 1 fixed and 2 and 8 programmable concurrent PMU events without multiplexing, respectively. This means that, when measuring more than the maximal concurrent events in a node, one need to be aware that the measurements will lack precision and may vary from one execution to another, even when running the same workload.

Challenge VII: Drawbacks of using multiple hardware counters

An experiment to evaluate the impact of concurrent measurements was executed in an i7 module. This experiment consists in running a deterministic algorithm several times while changing the number of monitored PMCs. The CPU cycles counter was chosen as a reference counter and the number of monitored counters was increased from 1 to 18. Each counter monitor is configured to collect data from all cores. For each run, the accumulated performance counter is computed and a box-plot of 25 samples of each configuration is plotted. Fig. 14 shows the results of this experiment. Since the workload is deterministic, the number of CPU cycles is expected to be the same for all runs. However, the results show that, as the number of monitored counters increases, the precision decreases. A similar behavior can be noticed with the second monitored variable (branch misses), showing a general behavior in all monitored programmable counters. It must be noticed that all the works in the literature modeling the performance (or the power consumption) do not take this aspect in consideration. Moreover, this is a crucial aspect to deal when using machine learning techniques to approximate functions, since they are deeply dependent on the data and can only make proper estimations when the range of the inputs are kept the same. ML needs to explore the search space, so it needs the maximal number of variables in order to select the most important ones, but as the number of monitored PMCs impacts their

Table 4
Hardware performance events available in RECS i7 and Atom modules.

Hardware event counter	Processor		Hardware event counter	Processor	
	Atom	i7		Atom	i7
instructions	Fixed	Fixed	L1-icache-prefetches	N/A	N/A
cycles	Prog.	Prog.	L1-icache-prefetch-misses	N/A	N/A
cache-references	Prog.	Prog.	LLC-loads	N/A	Prog.
cache-misses	Prog.	Prog.	LLC-load-misses	N/A	Prog.
branch-instructions	Prog.	Prog.	LLC-stores	N/A	Prog.
branch-misses	Prog.	Prog.	LLC-store-misses	N/A	Prog.
bus-cycles	Prog.	Prog.	LLC-prefetches	N/A	Prog.
idle-cycles-frontend	N/A	Prog.	LLC-prefetch-misses	N/A	Prog.
idle-cycles-backend	N/A	Prog.	dTLB-loads	N/A	Prog.
L1-dcache-loads	N/A	Prog.	dTLB-load-misses	N/A	Prog.
L1-dcache-load-misses	N/A	Prog.	dTLB-stores	N/A	Prog.
L1-dcache-stores	N/A	Prog.	dTLB-store-misses	N/A	Prog.
L1-dcache-store-misses	N/A	Prog.	dTLB-prefetches	N/A	N/A
L1-dcache-prefetches	N/A	N/A	dTLB-prefetch-misses	N/A	N/A
L1-dcache-prefetch-misses	N/A	Prog.	iTLB-loads	N/A	Prog.
L1-icache-loads	N/A	N/A	iTLB-load-misses	N/A	Prog.
L1-icache-load-misses	N/A	Prog.			

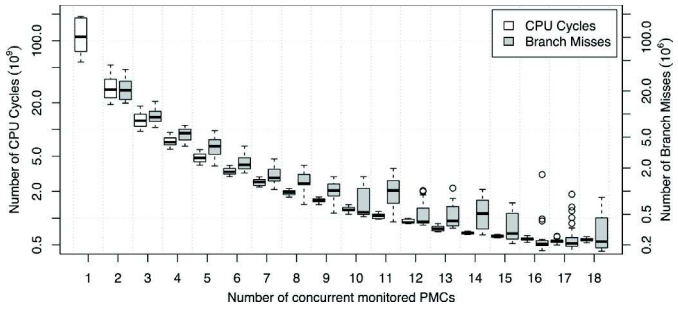


Fig. 14. Impact of the number of concurrent monitoring on the precision of the CPU cycles and Branch misses counters.

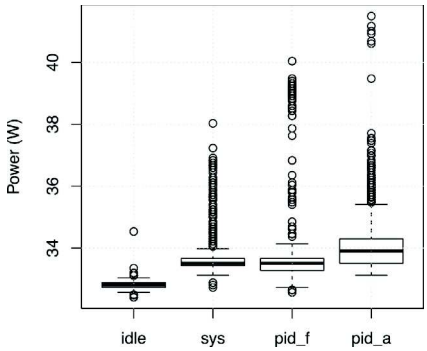


Fig. 15. Power impact of logging KPI at system-level only (`sys`), system and process-level with and without filtering (`pid_f`, `pid_a`).

accuracy, once the variables are selected, a second execution of the data acquisition must be done to reduce the issue of concurrent monitoring of PMCs. This aspect needs to be taken into account when implementing any model construction methodology.

Lesson VII: Reduce the number of monitored hardware counters to the minimum

5.4. Power consumption overhead

Challenge VIII: What is the power impact of the measure itself?

The monitoring of performance counters increases the power consumption of the system, since it increases system's overhead. This overhead is related to the quantity of monitored KPIs and their frequency of updates. An experiment was conducted to analyze the power impact of monitoring variables in an i7 module. Four scenarios were monitored during 1000 s with a sampling rate of 1 Hz, while using Linux on-demand governor. The `idle` scenario collected only machine's idle power with no KPI monitoring on the node; `sys` collected only KPIs at system-level; and `pid_f` and `pid_a` collected KPIs for system and process-level with and without filtering. The filter works by only measuring KPIs of the processes which executed at least one CPU cycle during the last 5 time steps, decreasing the total number of monitored processes from 157 to 70 processes while monitoring approximately 20 concurrent processes.

Fig. 15 shows that, when logging all implemented KPIs (around 240 monitored variables), the power consumption of the system increases according to the type of measurement done. System level measurements can increase up to 0.87 W (from 32.6 W on Idle to 33.5 W), while process level may make it larger going up to 0.95 W when filtered and even 1.31 W logging all process running in the operating system. As one can see the increase in power consumption is quite small when compared to the idle power, i.e. less than 4% in the worst case. It is important to notice that these power measurements consider the worst case scenario because they are done comparing to an idle system, which usually is not intended to be monitored. Power consumption of processors are not linear and present a significant jump from idle to active mode, mainly when using the on-demand governor.

Lesson VIII: The impact of measurements is not only higher power consumption but also reduced accuracy

6. Summary and conclusion

This article described issues of power and performance measuring. When using external power meters, the contributions on the total power of a PSU can be at the same order of magnitude or even higher than the reported accuracy of the state of the art models of power consumption in function of system values. This enhances the need of modeling its power losses in order to reach more accurate models and can be done using a linear regression model. Other power metering issues such as timing jitter and repeated values were identified and methodology of detection were discussed. On the performance measuring, several details need to be taken into account. The same property, such as processor's core frequency, can be measured in different ways, providing different accuracies. In addition, concurrent measurements of PMCs have a big impact on the overall counts and may impact models' estimations.

All of these aspects need to be carefully addressed when defining the scope of measured values. The number of measured values should be kept to a minimum, not only to reduce the performance impact on servers, but also to increase accuracy of measured values. In [15], we shown for instance that when using the measured values to model the power consumption of an infrastructure through machine learning, the accuracy and the precision are both increased significantly, up to 20% when combining PSU losses, jitter, repeated values and correct usage of PMC.

The measurements techniques presented in this work were used in our environment but can be easily generalized for any infrastructure. The PSU power modeling can be done either based on the vendor information, or using an external meter to measure the input power and a clamp meter to measure its output power under different power loads, providing the input data to create the PSU's losses model. All other techniques presented earlier are machine independent, and can be used for any architecture.

Acknowledgements

This research was supported in part by the European Commission under contract 288701 through the project CoolEmAll.

References

[1] American Power Conversion Corp, Metered Rack Power Distribution Unit, 2006.
[2] D. Bedard, M.Y. Lim, R. Fowler, A. Porterfield, PowerMon: fine-grained and integrated power monitoring for commodity computer systems, in: IEEE SoutheastCon, IEEE, 2010, pp. 479–484.
[3] H. Chen, W. Shi, Power measuring and profiling: the state of art, in: Handbook of Energy-Aware and Green Computing, 1st ed., Chapman & Hall/CRC, 2012.
[4] L.F. Cupertino, A command-line tool displaying power and energy consumption of each process. Deliverable d5.2, CoolEmAll project, 2012, September.
[5] L.F. Cupertino, G. Da Costa, A. Sayah, J.-M. Pierson, Energy consumption library, in: J.-M. Pierson, G. Da Costa, L. Dittmann (Eds.), Energy Efficiency in Large Scale Distributed Systems, Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2013, pp. 51–57.
[6] G. Da Costa, H. Hlavacs, Methodology of measurement for energy consumption of applications, 2010 11th IEEE/ACM International Conference on Grid Computing (GRID) (2010, October) 290–297.
[7] Dell Inc, PowerEdge M1000e Technical Guide, 2010, June.
[8] Eaton Corp, Advanced Enclosure Power Distribution Unit (ePDU) User's Guide, 2012.
[9] J. Edge, Perfcounters Added to the Mainline, LWN, 2009, July.
[10] Electronic Educational Devices Inc, Watts Up? and Watts Up? PRO Operators Manual, 2008.
[11] Energy Optimizers Ltd, Plogg – the Smart Energy Meter Plug: Terminal Software User Guide, 2007.
[12] Fermilab Scientific Linux. http://scientificlinux.org/.
[13] M.A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, J. Visser, SEFLab: a lab for measuring software energy footprints, 2013 2nd International Workshop on Green and Sustainable Software (GREENS) (2013, May) 30–37.
[14] J. Flinn, M. Satyanarayanan, PowerScope: a tool for profiling the energy usage of mobile applications, in: Proceedings. WMCSA '99. Second IEEE Workshop on Mobile Computing Systems and Applications, 1999, 1999, February, pp. 2–10.
[15] L.F. Cupertino, Modeling the Power Consumption of Computing Systems and Applications Through Machine Learning Techniques, University Paul Sabatier, Toulouse, France, 2015, July (PhD dissertation).
[16] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K.W. Cameron, Powerpack: energy profiling and analysis of high-performance systems and applications, IEEE Trans. Parallel Distrib. Syst. 21 (May (5)) (2010) 658–671.
[17] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, R. Geyer, An energy efficiency feature survey of the intel haswell processor, in: Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPSW '15, Washington, DC, USA, 2015, pp. 896–904, IEEE Computer Society.
[18] Intel Corp, Voltage Regulator-Down VRD 11.1, Processor Power Delivery Design Guidelines, Septembe, 2009.
[19] Intel Corp, Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3 (3A, 3B & 3C): System Programming Guide, 2014, June.
[20] M. Jarus, A. Oleksiak, T. Piontek, J. Węglarz, Runtime power usage estimation of hpc servers for various classes of real-life applications, Future Gener. Comput. Syst. 36 (2014) 299–310.
[21] G. Langeveld, netatop, 2013, July http://www.atoptool.nl/netatop.php.
[22] L.A. Liikkanen, T. Nieminen, Comparison of end-user electric power meters for accuracy. Tech report, Helsinki Institute for Information Technology, 2009, July.
[23] Linux, User's Manual. perf_event_open(2), 2015, January.
[24] J. Mair, Z. Huang, D. Eyers, L.F. Cupertino, G. Da Costa, J.-M. Pierson, H. Hlavacs, Power modeling, in: J.-M. Pierson (Ed.), Large-Scale Distributed Systems and Energy Efficiency: A Holistic View, John Wiley and Sons, 2015, April (chapter 5).
[25] J.C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A.C. Snoeren, R.K. Gupta, Evaluating the effectiveness of model-based power characterization, in: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11, Berkeley, CA, USA. USENIX Association, 2011, p. 12.
[26] D. Meisner, B.T. Gold, T.F. Wenisch, Powernap: eliminating server idle power, SIGARCH Comput. Archit. News 37 (1) (2009) 205–216.
[27] NVIDIA Corp, NVML API Reference Manual, 2013.
[28] P3 International Corp, Model P4482: Kill A Watt Control Operation Manual, 2013.
[29] E. Volk, W. Piatek, M. Jarus, G. Da Costa, L. Sisó, M. vor dem Berge, Update on definition of the hardware and software models. Tech report, CoolEmAll, 2013, May.
[30] M. vor dem Berge, J. Buchholz, L. Cupertino, G. Da Costa, A. Donoghue, G. Gallizo, M. Jarus, L. Lopez, A. Oleksiak, W. Piatek, E. Pages, J.-M. Pierson, T. Piontek, D. Rathgeb, J. Salom, L. Sisó, E. Volk, U. Wössner, T. Zilio, Coolemall: models and tools for planning and operating energy efficient data centres, in: S.U. Khan, A.Y. Zomaya (Eds.), Handbook on Data Centers, Springer New York, 2015, pp. 191–245.
[31] V.M. Weaver, Linux perf_event features and overhead, in: The 2nd International Workshop on Performance Analysis of Workload Optimized Systems, FastPath, 2013, April.
[32] V.M. Weaver, D. Terpstra, S. Moore, Non-determinism and overcount on modern hardware performance counter implementations, in: International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2013, pp. 215–224.
[33] H. Yang, Q. Zhao, Z. Luan, D. Qian, imeter: an integrated vm power model based on performance profiling, Future Gener. Comput. Syst. 36 (2014) 267–286.