

SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers

Guillaume Fieni, Romain Rouvoy, Lionel Seinturier

▶ **To cite this version:**

Guillaume Fieni, Romain Rouvoy, Lionel Seinturier. SmartWatts: Self-Calibrating Software-Defined Power Meter for Containers. CCGRID 2020 - 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May 2020, Melbourne, Australia. 10.1109/CCGrid49817.2020.00-45 . hal-02470128

HAL Id: hal-02470128
https://inria.hal.science/hal-02470128

Submitted on 23 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SMARTWATTS: Self-Calibrating Software-Defined Power Meter for Containers

Guillaume Fieni, Romain Rouvoy*, Lionel Seinturier
*Univ. Lille / Inria / *IUF*
firstname.lastname@univ-lille.fr

***Abstract*—Fine-grained power monitoring of software activities becomes unavoidable to maximize the power usage efficiency of data centers. In particular, achieving an optimal scheduling of containers requires the deployment of software-defined power meters to go beyond the granularity of hardware power monitoring sensors, such as *Power Distribution Units* (PDU) or Intel's *Running Average Power Limit* (RAPL), to deliver power estimations of activities at the granularity of software containers. However, the definition of the underlying power models that estimate the power consumption remains a long and fragile process that is tightly coupled to the host machine.**

To overcome these limitations, this paper introduces SMARTWATTS: a lightweight power monitoring system that adopts online calibration to automatically adjust the CPU and DRAM power models in order to maximize the accuracy of runtime power estimations of containers. Unlike state-of-the-art techniques, SMARTWATTS does not require any *a priori* training phase or hardware equipment to configure the power models and can therefore be deployed on a wide range of machines including the latest power optimizations, at no cost.

***Index Terms*—Energy, Containers, Power model**

I. INTRODUCTION

Modern data centers are continuously trying to maximize the *power usage efficiency* (PUE) of their hardware and software infrastructures to reduce their operating cost and eventually their carbon emission. While physical power meters offer a suitable solution to monitor the power consumption of physical servers, they fail to support the energy profiling at a finer granularity: dealing with the software services that are distributed across such infrastructures. To overcome this limitation, software-defined power meters build on power models to estimate the power consumption of software artifacts in order to identify potential energy hotspots and leaks in software systems [1] or improve the management of resources [2]. However, existing software-defined power meters are integrating power models that are statically designed, or learned prior to any deployment in production [3], [4]. This may result in inaccuracies in power estimations when facing unforeseen environments or workloads, thus affecting the exploitation process. As many distributed infrastructures, such as clusters or data centers, have to deal with the scheduling of unforeseen jobs, in particular when handling black-box virtual machines, we can conclude that the adoption of such static power models [5] has to be considered as inadequate in production. We therefore believe that the state-of-the-art in this domain should move towards the integration of more dynamic power models that can adjust themselves at runtime to better reflect the variation of the underlying workloads and to cope with the potential heterogeneity of the host machines.

In this paper, we therefore introduce SMARTWATTS, as a self-calibrating software-defined power meter that can automatically adjust its CPU and DRAM power models to meet the power accuracy requirements of monitored software containers. Our approach builds on the principles of sequential learning principles and proposes to exploit coarse-grained power monitors like *Running Power Average Limit* (RAPL), which is commonly available on modern Intel's and AMD's micro-architecture generations, to control the estimation error. We have implemented SMARTWATTS as an open source power meter to integrate our self-calibrating approach, which is in charge of automatically adjusting the power model whenever some deviation from the ground truth is detected. When triggered, the computation of a new power model aggregates the past performance metrics from all the deployed containers to infer a more accurate power model and to seamlessly update the software-defined power meter configuration, without any interruption. The deployment of SMARTWATTS in various environments, ranging from private clouds to distributed HPC clusters, demonstrates that SMARTWATTS can ensure accurate real-time power estimations (less than 3.5 % of error on average, at a frequency of 2 Hz) at the granularity of processes, containers and virtual machines. Interestingly, the introduction of sequential learning in software-defined power meters eliminates the learning phase, which usually last from minutes to hours or days, depending on the complexity of the hosting infrastructure [3], [5].

Additionally, our software-defined approach does not require any specific hardware investment as SMARTWATTS can build upon embedded power sensors, like RAPL, whenever they are available. The code of SMARTWATTS is made available online as open-source software[1] to encourage its deployment at scale and to leverage the adoption and reproduction of our results. The key contributions of this paper can therefore be summarized as follows:

1) a self-calibrating power modelling approach,
2) CPU & DRAM models supporting power states,
3) an open source implementation of our approach,
4) an assessment on container-based environments.

In the remainder of this paper, we start by providing some background on state-of-the-art power models and their

[1] https://github.com/powerapi-ng/smartwatts-formula

limitations (cf. Section II) prior to introducing our contribution (cf. Section III). Then, we detail the implementation of SMARTWATTS as an extension of the BITWATTS middleware framework (cf. Section IV) and we assess its validity on three scenarios (cf. Section V). We conclude and provide some perspectives for this work in Section VI.

II. RELATED WORK

A. Hardware Power Meters

Over the years, hardware power meters have evolved to deliver hardware-level power measurements with different levels of granularity, from physical machines to electronic components.

WATTPROF [6] power monitoring platform supports the profiling of *High Performance Computing* (HPC) applications. This solution is based on a custom board, which can collect raw power measurements from various hardware components (CPU, disk, memory, etc.) from sensors connected to power lines. The board can connect up to 128 sensors that can be sampled at up to $12\,KHz$. As in [7], the authors argue that this solution is able to perform per-process power estimation, but they only validate their approach while running a single application.

WATTWATCHER [4] is a tool that can characterize workload energy power consumption. The authors use several calibration phases to build a power model that fits a CPU architecture. This power model uses a predefined set of *Hardware Performance Counters* (HWPC) as input parameters. As the authors use a special power model generator that can target any CPU architecture, which has be to carefully described.

RAPL [8] offers specific *hardware performance counters* (HWPC) to report on the energy consumption of the CPU since the "Sandy Bridge" micro-architecture for Intel (2011) and "Zen" for AMD (2017). Intel divides the system into domains (PP0, PP1, PKG, DRAM) that report the energy consumption according to the requested context. The PP0 domain represents the core activity of the processor (cores + L1 + L2 + L3), the PP1 domain the uncore activities (LLC, integrated graphic cards, etc.), and PKG represents the sum of PP0 and PP1, and the DRAM domain exhibits the DRAM energy consumption. Desrochers *et al.* demonstrate the accuracy of the DRAM power estimations of RAPL, especially on Intel Xeon processors [9].

B. Software-Defined Power Meters

To get rid of the hardware cost imposed by the above solutions, the design of power models has been regularly considered by the research community over the last decade, in particular for CPU [5], [10]–[13]. Notably, as most architectures do not provide fine-grained power measurement capabilities, McCullough *et al.* [12] argue that power models are the first step towards enabling dynamic power management for power proportionality at all levels of a system.

While standard operating system metrics (CPU, memory, disk, or network), directly computed by the kernel, tend to exhibit a large error rate due to their lack of precision [11], [13], HWPC can be directly gathered from the processor (*e.g.*, number of retired instructions, cache misses, non-halted cycles). Modern processors provide a variable number of HWPC events, depending on the generation of the micro-architectures and the model of the CPU. As shown by Bellosa [10] and Bircher [14], some HWPC events are highly correlated with the processor power consumption, while the authors in [15] concluded that not all HPC are relevant, as they may not be directly correlated with dynamic power.

Power modeling often builds on these raw metrics to apply learning techniques [16] to correlate the metrics with hardware power measurements using various regression models, which are so far mostly linear [12]. Three key components are commonly considered to train a power model: *a)* the workload(s) to run during sampling, *b)* the minimal set of input parameters, and *c)* the class of regression to use [16]–[19].

The workloads used along the training phase have to be carefully selected to capture the targeted system. In this domain, many benchmarks have been considered, but they are mostly *a)* designed for a given architecture [16], [20], *b)* manually selected [5], [17]–[19], [21]–[24], or even *c)* private [17]. Unfortunately, this often leads to the design of power models that are tailored to a given processor architecture and manually tuned (for a limited set of power-aware features) [16], [17], [20], [21], [23]–[26].

C. Limitations & Opportunities

To the best of our knowledge, the state of the art in hardware power meters often imposes hardware investments to provide power measurements with an high accuracy, but a coarse granularity, while software-defined power meters target fine-grained power monitoring, but often fail to reach high accuracy on any architecture and/or workload.

This paper clearly differs from the state of the art by providing an open source, modular, and self-adaptive implementation of a self-calibrating software-defined power meter: SMARTWATTS. As far as we know, our implementation is the first to deliver both CPU and DRAM power estimations at runtime for any software packaged as processes, containers or virtual machines. Unlike existing approaches published in the literature, the approach we describe is *i)* architecture agnostic, *ii)* processor aware, and *iii)* dynamic. So far, the state of the art fails to deploy software-defined power meters in productions because *i)* the model learning phase can last from minutes to days, *ii)* the power models are often bound to a specific context of execution that do not take into account hardware energy-optimization states, and *iii)* the reference power measurement requires specific hardware to be installed on a large amount of nodes. This therefore calls for methods that can automatically adapt to the hardware and workload diversities of heterogenous environments in order to maintain the accuracy of power measurements at scale.

III. SMARTWATTS POWER MONITORING

We therefore propose to support self-calibrating power models that leverage *Reference Measurements* and *Hardware Per-*

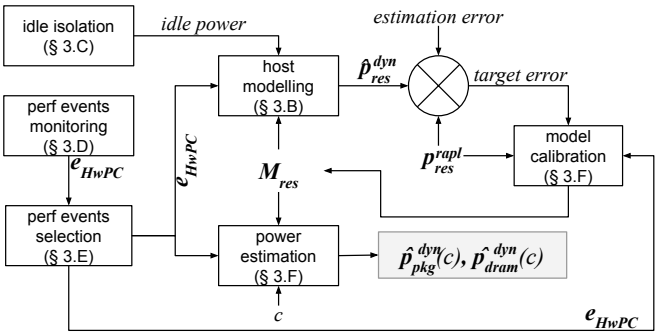


Fig. 1. Overview of SMARTWATTS

formance Counters (HWPC) to estimate the power consumption at the granularity of software containers along multiple resources: CPU and DRAM. More specifically, our contribution builds upon two widely available system interfaces: *RAPL* to collect baseline measurements for CPU and DRAM power consumptions, as well as Linux's *perf_events* interface to capture the *Hardware Performance Counters* (HWPC) events used to estimate the per-container power consumption from resource-specific power models, which are adjusted at runtime.

A. Overview of SMARTWATTS

Figure 1 introduces the general architecture of SMARTWATTS. SMARTWATTS manages at runtime a set of self-calibrated power models (M^f_{res}) for each *power-monitorable* resource res (*e.g.*, CPU, DRAM). These power models are then used by SMARTWATTS to estimate the power consumptions of *i)* the host $\hat{p}_{res}$ and *ii)* all the hosted containers c: $\hat{p}_{res}(c)$.

SMARTWATTS uses $\hat{p}_{res}$ to continuously assess the accuracy of the managed power models (M^f_{res}) and to ensure that the estimated power consumption does not diverge from the baseline measurements reported by RAPL (p^{rapl}_{res}, cf. Section III-B). Whenever the estimated power consumption error (ϵ_{res}) diverges from the baseline measurements beyond a configured threshold, SMARTWATTS automatically triggers a new online calibration process of the diverging power model to better match the current input workload.

To better capture the dynamic power consumption of the host, SMARTWATTS needs to isolate its static consumption. To do so, we use a dedicated component that activates when the machine is at rest—*e.g.*, after booting (cf. Section III-C)—to monitor the power activity of the host.

In addition to the static constant, SMARTWATTS estimates the power consumption of the host from a set of raw input values that refers to HWPC events, which are selected *at runtime* (cf. Section III-E).

This design ensures that SMARTWATTS keeps adjusting its power models to maximize the accuracy of power estimations. Therefore, unlike the state-of-the-art power monitoring solutions, SMARTWATTS does not suffers from estimation errors due to the adoption of an inappropriate power model as it autonomously optimizes the underlying power model whenever a accuracy anomaly is detected.

B. Modelling the Host Power Consumption

For each resource $res \in \{pkg, dram\}$ exposed by the RAPL interface, the associated power consumption p^{rapl}_{res} can be modelled as:

$$p^{rapl}_{res} = p^{static}_{res} + p^{dyn}_{res} \quad (1)$$

where p^{static}_{res} refers to the static power consumption of the monitored resource (cf. Section III-C), and p^{dyn}_{res} reflects the dynamic power dissipated by the processor along the sampling period.

Then, we can compute a power model $M^f_{res} = [\alpha_0, \cdots, \alpha_n]$ that correlates, for a given frequency f (among available frequencies F, cf. Section III-D), the dynamic power consumption ($\hat{p}^{dyn}_{res}$) to the raw metrics reported by a set of of *Hardware Performance Counter* (HwPC) events (cf. Section III-E), $E^f_{res} = [e_0, \ldots, e_n]$:

$$\exists f \in F,\ \hat{p}^{dyn}_{res} = M^f_{res} \cdot E^f_{res} \quad (2)$$

We build M^f_{res} from an *Elastic net* regression—that linearly combines the L1 and L2 penalties of the *Lasso* and *Ridge* methods—applied on the past k samples $S^f_k = \langle p^{dyn}_{res}, E^f_{res} \rangle$, with $p^{dyn}_{res} = p^{rapl}_{res} - p^{static}_{res}$. To ensure that the power consumption of every single container is linear with regards to the global power consumption of the node, we enforce the inference of positive coefficients when computing the regression and we check that the intercept belongs to the range $[0, \text{TDP}]$ where TDP refers to the *Thermal Design Power* of the CPU. By comparing $p^{dyn}_{res} + p^{static}_{res}$ with p^{rapl}_{res}, we can continuously estimate the error $\varepsilon_{res} = \mid p^{dyn}_{res} - \hat{p}^{dyn}_{res} \mid$ from estimated values in order to monitor the accuracy of the power model M^f_{res}. Whenever the error exceeds a given threshold set by the administrator, a new power model is generated for the frequency f by integrating the latest samples.

C. Isolating the Static Power Consumption

Isolating the static power consumption of a node is a challenging issue as it requires to reach a quiescient state in order to capture the power consumption of the host at rest. To capture this information, we designed and implemented a power logger component that runs as a lightweight daemon with low priority that periodically logs the package and DRAM power consumptions reported by RAPL. Then, we compute the *median* value and the *interquartile range* (IQR) from gathered measurements to define the p^{static}_{res} constant as : $p^{static}_{res} = median_{res} - 1.5 \times IQR_{res}$. This approach intends to filter out outliers reported by RAPL, including periodic measurement errors we observed, and to consider the lowest power consumption observed along a given period of time as the static consumption of a node.

By default, SMARTWATTS assumes that the static consumption of the host does not requires to be spread across the active containers. However, other power accounting policies can be implemented. For example, by reporting an empty static

consumption, SMARTWATTS will share it across the running containers depending on their activity.

D. Monitoring Power States & HWPC Events

As previously introduced, the accuracy of a power model M^f_{res} strongly depends on *i)* the selection of relevant input features (HWPC events e_n) and *ii)* the acquisition of input values that are evenly distributed along the reference power consumption range. This is one of the reasons why the input workloads used in standard calibration phases are often critical to capture an accurate power model that reflects the power consumption of a host for a given class of applications. SMARTWATTS rather promotes a self-calibrating approach that does not impose the choice of a specific benchmark or workload, but exploits the ongoing activity variations of the host machine to continuously adjust its power models. To achieve this, SMARTWATTS monitors selected sets of HWPC events and stores the associated samples in memory. To better deal with the power features of hardware components, we group the input samples per operating frequency. This allows to calibrate frequency-specific power models when an estimation arises, with the goal to converge automatically to a stable and precise power model over the time.

By balancing the samples along the range of frequencies operated by the processor, SMARTWATTS ensures that the power model learning phase does not overfit the current context of execution, which may lead to the generation of unstable power models, thus impacting the accuracy of the power measurements. The sampling tuples S^f_k are grouped into memory as frequency layers $L^f_{res} = [S^f_0, ..., S^f_n]$, which are the raw features we maintain to build M^f_{res}.

To store the samples in the layer corresponding to the current frequency of the processor, SMARTWATTS compute the average running frequency as follows:

$$F_{avg} = F_{base} * \frac{\Delta \texttt{ APERF}}{\Delta \texttt{ MPERF}} \tag{3}$$

where F_{base} is the processor base frequency constant extracted from the *Model Specific Registers* (MSR) `PLATFORM_INFO`. `APERF` and `MPERF` are MSR-based counters that increment at the current and maximum frequencies, respectively. These counters are continuously updated, hence they report on a precise average frequency without consuming the limited HWPC slots. Interestingly, the performance power states, such as P-states and Turbo Boost, will be accounted by these counters as they act mainly on the frequency of the core in order to boost the performance. The idle optimization states (C-states) will also be accounted as they mainly reduce of the average frequency of the core towards its *Max Efficiency Frequency* before being powered-down.

E. Selecting the Correlated HWPC Events

The second challenge of SMARTWATTS consists in selecting at runtime the relevant HWPC events that can be exploited to accurately estimate the power consumption. To do so, we list the available events exposed by the host's *Performance Monitoring Units* (PMU) and we evaluate their correlation with the power consumption reported by RAPL. Instead of testing each available HWPC events, we narrow the search using the PMU associated to the modelled component—*i.e.*, we consider the HWPC events from the core PMU to model the PKG power consumption. As reference events, we consider `unhalted-cycles` for the package and `llc-misses` for the DRAM, which are the standard HWPC events available across many processor architectures, and have been widely used by the state of the art to design power models [3]–[5]. To elect a HWPC event as a candidate for the power model, we first compute the Pearson coefficient $r_{e,p}$ for n values reported by each monitored HWPC event e and the base power consumption p reported by RAPL:

$$r_{e,p} = \frac{\sum_{i=1}^{n} (e_i - \overline{e})\ (p_i - \overline{p})}{\sqrt{\sum_{i=1}^{n} (e_i - \overline{e})^2}\ \sqrt{\sum_{i=1}^{n} (p_i - \overline{p})^2}} \tag{4}$$

Then, SMARTWATTS stores the list of HWPC events that exhibit a better correlation coefficient r than the baseline event for DRAM and PKG. This list of elected HWPC events is further used as input features to implement the PKG and DRAM power models exploited by SMARTWATTS.

F. Estimating the Container Power Consumption

Given that we learn the power model M^f_{res} from aggregated events, $E^f_{res} = \sum_{c \in C} E^f_{res}(c)$, we can predict the power consumption of any container c by applying the inferred power model M^f_{res} at the scale of the container's events $E^f_{res}(c)$:

$$\exists f \in F,\ \forall c \in C,\ \hat{p}^{dyn}_{res}(c) = M^f_{res} \cdot E^f_{res}(c) \tag{5}$$

Then, we distribute the value of the intercept i that is included in the estimate $\hat{p}^{dyn}_{res}(c)$ proportionally to the dynamic part of the consumption of c

$$\forall c \in C,\ \tilde{p}^{dyn}_{res}(c) = \hat{p}^{dyn}_{res}(c) - i \times (1 - \frac{\hat{p}^{dyn}_{res}(c) - i}{\hat{p}^{dyn}_{res} - i}) \tag{6}$$

In theory, one can expect that $\hat{p}^{dyn}_{res} \stackrel{!}{=} p^{dyn}_{res}$ if the model perfectly estimates the dynamic power consumption but, in practice, the predicted value may introduce an error $\varepsilon_{res} = | p^{dyn}_{res} - \hat{p}^{dyn}_{res} |$. Therefore, we cap the power consumption of any container c as:

$$\forall c \in C,\ \lceil \tilde{p}^{dyn}_{res}(c) \rceil = \frac{p^{dyn}_{res} \times \tilde{p}^{dyn}_{res}(c)}{\hat{p}^{dyn}_{res}} \tag{7}$$

to ensure that $p^{dyn}_{res} = \sum_{c \in C} \lceil \tilde{p}^{dyn}_{res}(c) \rceil$, thus avoiding potential outliers. Thanks to this approach, we can also report on the confidence interval of the power consumption of containers by scaling down the observed global error:

$$\forall c \in C,\ \varepsilon_{res}(c) = \frac{\tilde{p}^{dyn}_{res}(c)}{\hat{p}^{dyn}_{res}} \times \varepsilon_{res} \tag{8}$$

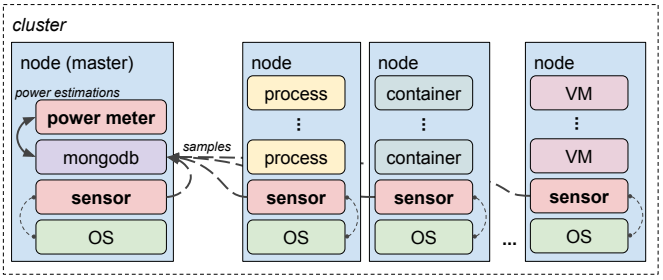


Fig. 2. Deployment of SMARTWATTS

In the following sections, we derive and implement the above formula to report on the power consumption of pkg and $dram$ resources. Our empirical evaluations report on the capped power consumptions for pkg ($\lceil \hat{p}_{pkg}^{dyn} \rceil$) and $dram$ ($\lceil \hat{p}_{dram}^{dyn} \rceil$), as well as the associated errors ε_{pkg} and ε_{dram}, respectively.

IV. IMPLEMENTATION OF SMARTWATTS

We implemented SMARTWATTS as a modular software system that can run atop a wide diversity of production environments. As depicted in Figure 2, our open source implementation of SMARTWATTS mostly rely on 2 software components—a sensor and a power meter—which are connected with a MONGODB database.[2] MONGODB offers a flexible and persistent buffer to store input metrics and power estimations. The sensor is designed as a lightweight process that is intended to run on target nodes with a limited impact. The power meter is a remote service that can be deployed whenever needed. SMARTWATTS uses this feature to support both online and *post mortem* power estimations, depending on use cases.

A. Client-side Sensor

The component sensor consists in a lightweight software daemon deployed on all the nodes that need to be monitored.

Static power isolation: When the node boots, the sensor starts the idle consumption isolation phase (cf. Section III-C) by monitoring the PKG and DRAM power consumptions reported by RAPL along the global idle CPU time and the `fork`, `exec` and `exit` process control activities provided by Linux *process information pseudo-filesystem* (procfs). Whenever a process control activity or the global idle CPU time exceed 99 % during this phase, the power samples are discarded to prevent the impact of background activities on the static power isolation process. As stated in III-C, this phase is only required when the idle attribution policy consider the idle consumption as a power leakage. It is not needed to run this phase as long as there is no change in the hardware configuration of the machine (specifically CPU or DRAM changes).

[2]https://www.mongodb.com

Event selection: Once completed, the sensor switches to the event selection phase (cf. Section III-E). To select the most accurate *Hardware performances counters* to estimate the power of a given node, SMARTWATTS need to identify the HWPC statistically correlated with the power consumption of the components. For that, the sensor monitors the power consumption reported by RAPL and the maximum simultaneous HWPC events possible without multiplexing, as it can a significant noise and distort the correlation coefficient of the events, over a (configurable) period of 30 ticks. The maximal amount of simultaneous HWPC events depends of the micro-architecture of the CPU and will be detected at runtime using the PMU detection feature of the *libpfm4* library.[3] We then correlate the power consumption with the values of the monitored HWPC events and rank them by highest correlation with RAPL and lowest correlation across the other HWPC. Whenever possible, fixed HWPC event counters are selected in priority to avoid consuming a programmable counter.

Control groups: SMARTWATTS leverages the *control groups* (Cgroups) implemented by Linux to support a wide range of monitoring granularities, from single processes, to software containers (DOCKER),[4] to virtual machines (using LIBVIRT).[5] The sensor also implement a kernel module that is in charge of configuring the Cgroups to monitor the power consumption of kernel and system activities, which is not supported by default. To do so, this module defines 2 dedicated Cgroups for the roots of the system and the kernel process hierarchy.

Event monitoring: Once done with the above preliminary phases, the sensor automatically starts to monitor the selected HWPC events together with RAPL measurements for the DRAM and CPU components at a given frequency and it reports these samples to the MONGODB backend (cf. Section III-D). The sensor monitors the selected HWPC events for the host and all the Cgroups synchronously to ensure that all the reported samples are consistent when computing the power models.

B. Server-side Power Meter

The power meter is implemented as a software service that requires to be deployed on a single node (*e.g.*, the master of a cluster). The power meter can be used online to produce real-time power estimations or offline to conduct *post mortem* analysis. This component consumes the input samples stored in the MONGODB database and produces power estimations accordingly. SMARTWATTS adopts a modular architecture based on the actor programming model, which we use to integrate a wide range of input/output data storage technologies (MongoDB, InfluxDB, etc.) and to implement power estimations at scale by devoting one actor per power model.

[3]http://perfmon2.sourceforge.net
[4]https://docker.com
[5]https://libvirt.org

Power modelling: The power meter provides an abstraction to build power models. In this paper, the power model we report on is handled by *Scikit-Learn*, which is the *de facto* standard Python library for general-purpose machine learning.[6] We embed the Ridge regression of *Scikit-Learn* in an actor, which is in charge of delivering a power estimation whenever a new sample is fetched from the database.

Model calibration: When the error reported by the power model exceeds the threshold defined by the user, the power meter triggers a new calibration of the power model to take into account the latest samples. This new power model is checked against the last sample to estimate its accuracy. If it estimates the power consumption below the configured threshold, then the actor is updated accordingly.

Power estimation: Power estimations are delivered at the scale of a node and for the Cgroups of interest. These scope of these Cgroups can reflect the activity of nodes' kernel and system, as well as any job or service running in the monitored environment. These power estimations can then be aggregated by owner, service identifier or any other key, depending on use cases. They can also be aggregated along time to report on the energy footprint of a given software system.

V. Validation of SmartWatts

This section assesses the efficiency and the accuracy of SmartWatts to evaluate the power consumption of running software containers.

A. Evaluation Methodology

We follow the experimental guidelines reported by [27] to enforce the quality of our results.

Testbeds & workloads: While our production-scale deployments of SmartWatts cover both Kubernetes and OpenStack clusters, for the purpose of this paper, we chose to report on more standard benchmarks, like Stress NG[7] and NASA's *NAS Parallel Benchmarks* (NPB) [28] to highlight the benefits of our approach.

Our setups are reproduced on the Grid5000 testbed infrastructure,[8] which provides multiple clusters composed of powerful nodes. In this evaluation, we use a Dell PowerEdge C6420 server having two Intel® Xeon® Gold 6130 Processors (Skylake) and 192 GB of memory (12 slots of 16 GB DDR4 2666MT/s RDIMMs). We are using the Ubuntu 18.04.3 LTS Linux distribution running with the 4.15.0-88-generic Kernel version, where only a minimal set of daemons are running in background. As stated in IV-A, we are using the Cgroups to monitor the activity of the running processes independently. In the case of the system services managed by systemd and the services running in Docker containers, their Cgroups membership is automatically handled as part of their lifetime management.

For this host, the reported TDP for the CPU is 125 Watts and 26 Watts for the DRAM. Theses values were obtained from the `PKG_POWER_INFO` and `DRAM_POWER_INFO` *Model Specific Registers* (MSR). The energy and performance optimization features of the CPU—*i.e.*, *Hardware P-States* (HWP), *Hyper-Threading* (HT), *Turbo Boost* (TB) and *C-states*, are fully enabled and use the default configuration of the distribution. The default CPU scaling driver and governor for the distribution are *intel_pstate* and *powersave*.

In all our experiments, we configure SmartWatts to report power measurements twice a second ($2\,Hz$) with an error threshold of 5 Watts for the PKG and 1 Watt for the DRAM.

Objectives: We evaluate SmartWatts with the following criteria:

- The *quality* of the power estimations when running sequential and parallel workloads;
- The *accuracy and stability* of the power models across different workloads;
- The *overhead* of the SmartWatts sensor component on the monitored host.

Reproducibility: For the sake of reproducible research, SmartWatts, the necessary tools, deployment scripts and resulting datasets are open-source and publicly available on GitHub.[9]

B. Experimental Results

Quality of estimations: Figure 3 first reports on the PKG and DRAM power consumptions we obtained with SmartWatts. The first line (rapl) refers to the ground truth power measurements we sample for the PKG and the DRAM via the HwPC events `RAPL_ENERGY_PKG` and `RAPL_ENERGY_DRAM`, respectively. The second line (global) refers to the power measurements estimated by SmartWatts for the PKG and the DRAM components from `CPU_CLK_THREAD_UNHALTED:REF_P` and `CPU_CLK_THREAD_UNHALTED:THREAD_P` fixed counters, `INSTRUCTIONS_RETIRED`, and `LLC_MISSES` programmable counters. The list of events has been automatically selected by the sensor component as presenting the best correlation with RAPL samples, as described in Section III-E. The error for each of the power models are further discussed in Figures 4 and 5.

The lines kernel and system isolates the power consumption induced by all kernel and system activities. Kernel activities include devices specific background operations, such as *Network interface controller* (NIC) and disks I/O processing queues, while system activities covers the different services, like the SSH server and Docker daemon, running on the node.

The remaining lines reports on individual power consumptions of a set of NPB benchmarks, which are executed in sequence (`lu`, `ep`, `ft`) or concurrently (`ft`, `cg`, `ep`, `lu`, `mg`) with variable number of cores (ranging from 8 to 32 cores). One can observe that SmartWatts supports the isolation of power consumptions at process-level by leveraging Linux

[6] https://scikit-learn.org
[7] https://launchpad.net/stress-ng
[8] https://www.grid5000.fr
[9] https://github.com/powerapi-ng/smartwatts-formula

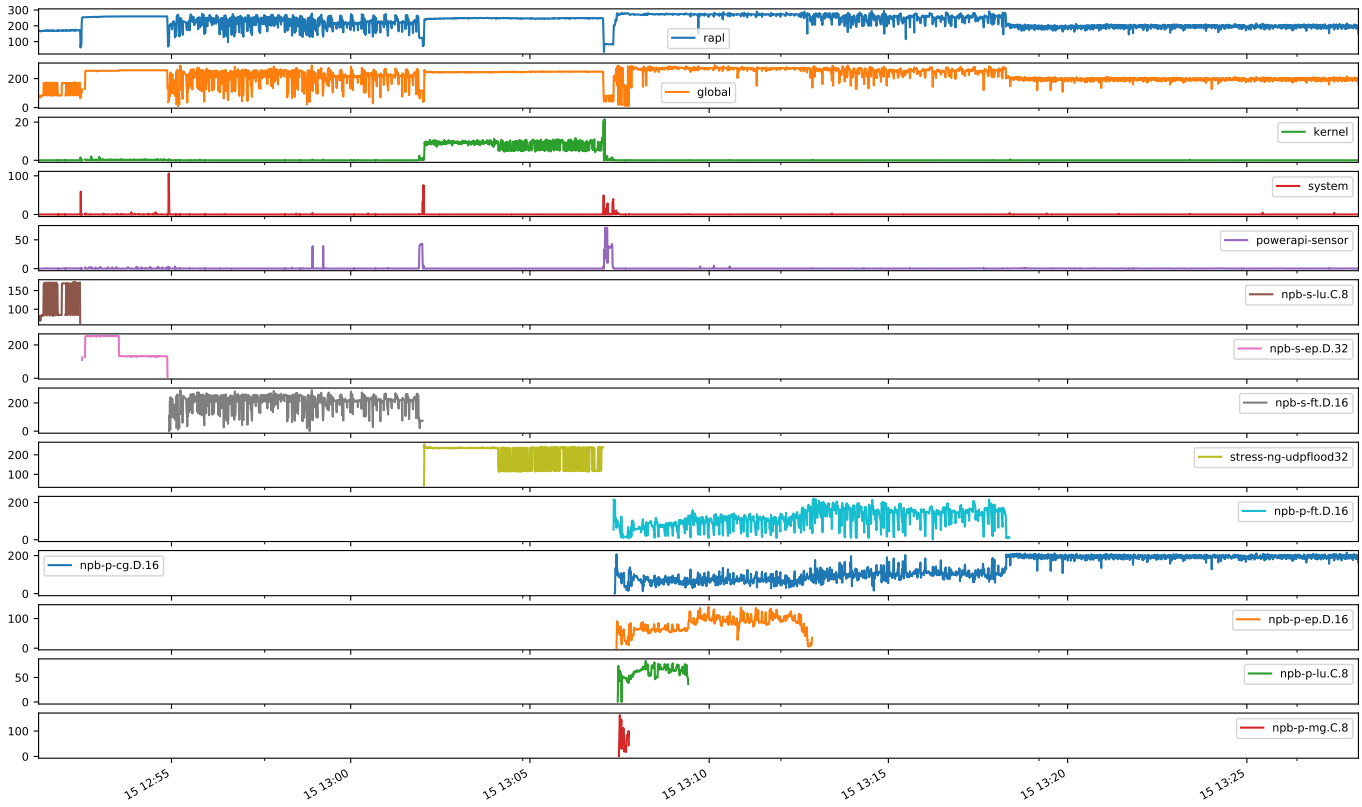


Fig. 3. Evolution of the aggregated PKG & DRAM power consumption (in Watts) along time and containers

Cgroups. This granularity allows SMARTWATTS to monitor indifferently processes, containers or virtual machines.

We also run `stress-ng` to observe potential side effects on the kernel activity by starting 32 workers that attempt to flood the host with UDP packets to random ports. While it remains negligible compared to the power consumption of the UDP flood process (3.26 W vs. 94.46 W on average), one can observe that this stress induces a lot of activity at the kernel to handle I/O, while the rest system is not severely impacted.

One can also observe that our sensor induces a negligible overhead (less than 0.2 Watts) with regards to the consumption of surrounding activities.

Estimation accuracy: Figures 4 and 5 reports on the distribution of estimation errors we observed per frequency and globally (right part of the plots) for the above scenario. We also report on the number of estimations produced for each of the frequency (upper part of the plots). While the error threshold for CPU and DRAM is set to 5 Watts and 1 Watts, one can observe that SMARTWATTS succeeds to estimate the power consumption with less than 3 Watts and 0.5 Watt of error for the PKG and DRAM components, respectively. The only case where estimation error grows beyond this threshold refers to the frequency 1000 Hz of the CPU (cf. Figures 4).The frequency 1000 Hz refers to the idle frequency of the node and the sporadic triggering of activities in this frequency induces a chaotic workload which is more difficult to capture for SMARTWATTS given the limited number of samples acquired

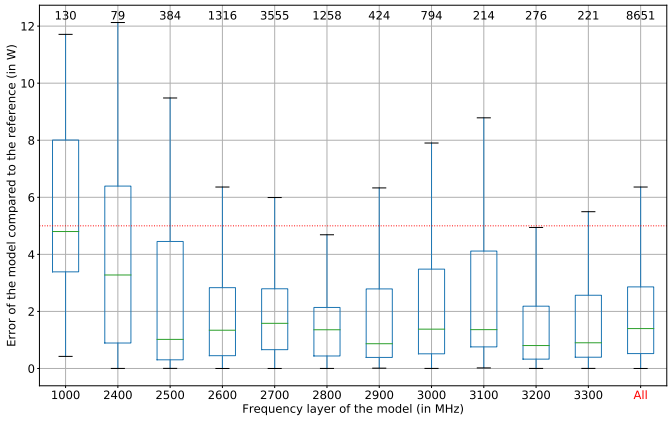


Fig. 4. Global & per-frequency error rate of the PKG power models

in this frequency (130 samples against 3555 samples for the frequency 2700 Hz).

The DRAM component, however, provides a more straightforward behavior to model with the selected HWPC events and therefore reports an excellent accuracy, no matter the operating frequency of the CPU package (cf. Figure 5).

The accuracy of the power models generated by SMARTWATTS are further detailed in Table I. While our approach succeeds to deliver accurate estimations of the power consumption for both CPU and DRAM components, the maximum error

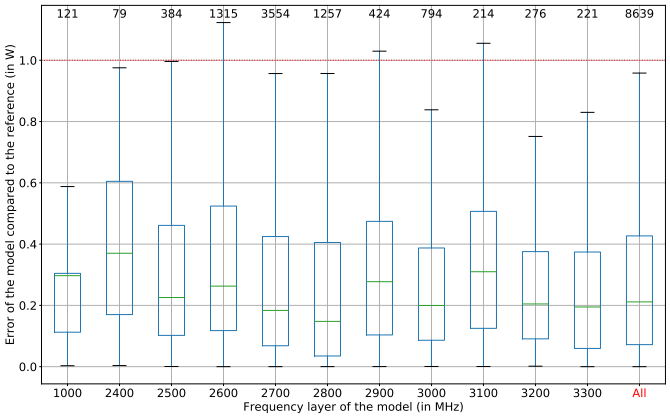


Fig. 5. Global & per-frequency error rate of the DRAM power models

refers to the bootstrapping phase of the sensor that requires to acquire a sufficiently representative number of samples in order to build a stable and accurate power model.

TABLE I
PER-SOCKET PKG & DRAM POWER MODELS ACCURACY

Resource	Socket	ε_{min}	ε_{max}	ε_{mean}	ε_{std}
PKG	0	0.000 W	117.563 W	2.625 W	5.548 W
	1	0.000 W	97.906 W	3.278 W	6.743 W
DRAM	0	0.000 W	26.114 W	0.465 W	0.839 W
	1	0.000 W	5.737 W	0.309 W	0.442 W

Model stability: Beyond the capability to accurately estimate the power consumption of software containers, we are also interested in assessing the capability of SMARTWATTS to generate stable power models over time. Tables II and III therefore reports, for each frequency, on metrics about the stability of power models. In particular, we look at the number of correct estimations produced by the power models in a given frequency. Given our input workloads, we can observe that SMARTWATTS succeeds to reuse a given power model up to 594 estimations, depending on frequencies. While we observed that the stability of our power models strongly depends on the sampling frequency, the error threshold, as well as the input workloads, one should note that the overhead for calibrating a power model in a given frequency does not take more than a couple milliseconds, which is perfectly acceptable when monitoring software systems in production.

Monitoring overhead: Regarding the runtime overhead of SMARTWATTS, one can observe in Figure 3 that the power consumption of SMARTWATTS is negligible compared to the hosted software containers. To estimate this overhead, we leverage the fact that the sensor component is running inside a software container, thus enabling SMARTWATTS to estimate its own power consumption. In particular, one can note in Table IV that the sensor power consumption represents 0.333 Watts for the PKG and 0.030 Watts for the DRAM, on average, when running at a frequency of $2\,Hz$.

TABLE II
PKG POWER MODELS STABILITY PER FREQUENCY

Frequency	models	total	min	max	mean	std
1000	130	130	1	13	1.710	1.881
2400	35	79	1	19	2.323	3.345
2500	72	384	1	58	3.764	10.226
2600	152	1316	1	417	6.045	29.904
2700	239	3555	1	358	8.610	27.591
2800	95	1258	1	294	9.530	39.320
2900	64	424	1	266	5.108	29.061
3000	134	794	1	121	4.783	13.581
3100	56	214	1	127	3.890	17.061
3200	53	276	1	115	5.307	17.817
3300	44	221	1	79	4.911	12.770

TABLE III
DRAM POWER MODELS STABILITY PER FREQUENCY

Frequency	models	total	min	max	mean	std
1000	21	121	1	58	6.050	14.009
2400	18	79	1	14	4.647	3.390
2500	82	384	1	129	4.425	14.082
2600	290	1315	1	70	3.887	7.961
2700	499	3554	1	594	4.980	25.637
2800	106	1257	1	300	7.618	37.541
2900	89	424	1	33	4.326	5.469
3000	137	794	1	60	5.293	8.270
3100	58	214	1	14	3.754	3.444
3200	50	276	1	62	5.520	9.361
3300	34	221	1	29	6.314	8.312

The usage of the *Hardware Performance Counters* (HwPC) is well known for its very low impact on the observed system, hence it does not induce runtime performance penalties [5], [19], [29], [30]. Additionally, we carefully took care of the cost of sampling these HwPC events and executing as little as possible instructions on the monitored nodes.

TABLE IV
PER-COMPONENT POWER CONSUMPTION OF THE SENSOR

Power	min	max	mean	std
PKG	0.0 W	63.961 W	0.333 W	3.097 W
DRAM	0.0 W	8.365 W	0.030 W	0.283 W

By proposing a lightweight and packaged software solution that can be easily deployed across monitored hosts, we facilitate the integration of power monitoring in large-scale computing infrastructures. Futhermore, the modular architecture of SMARTWATTS can accommodate existing monitoring infrastructures, like KUBERNETES METRICS or OPENSTACK CEILOMETER, to report on the power consumption of applications. The following section therefore demonstrates this capability by deploying a distributed case study atop of a KUBERNETES cluster.

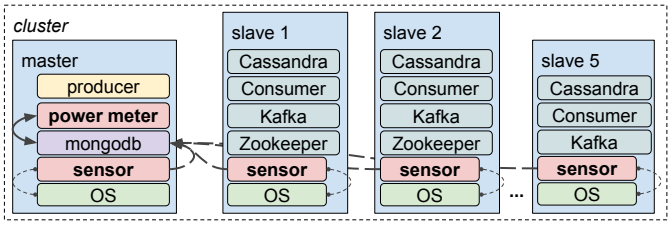


Fig. 6. Deployment of Kubernetes IoT backend services across 6 nodes

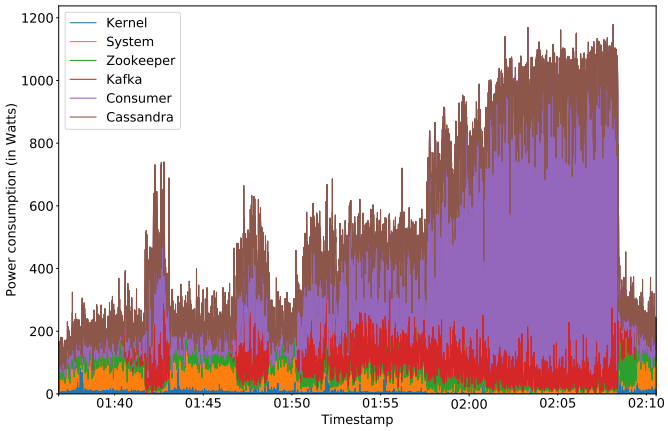


Fig. 7. Monitoring of service-level power consumptions

C. Tracking the Energy Consumption of Distributed Systems

To further illustrate the capabilities of SMARTWATTS, we take inspiration from [31] to deploy a distributed software systems that processes messages forwarded by IoT devices to a pipeline of processors connected by a KAFKA cluster to a CASSANDRA storage backend. Figure 6 depicts the deployment of this distributed system on a KUBERNETES cluster composed of 1 master and 5 slave nodes of the same configuration as stated in V-A. The input workload consists in a producer injecting messages in the cluster with a throughput ranging from 10 to 100 MB/s.

Figure 7 reports on the evolution of the power consumption per service while injecting the workload from the master node. One can observe that, when increasing the message throughput, the most impacted service is the Consumer, which requires extensive energy to process all the messages enqueued by the Kafka service. This saturation of the Consumer service seems to represent a core bottleneck in the application.

To further dive into this problem, we consider another perspective on the deployment in order to investigate the source of this efficiency limitation. While the execution of this workload requires 1.32 MJoules of energy to process the whole dataset, Figure 8 further dives inside the distribution of the energy consumption of individual pods along the PKG and DRAM components as a Sankey diagram [32]. This diagram builds on the capability of SMARTWATTS to aggregate power estimations along time to report on the energy consumption, as well as its capacity to track power consumption from software processes (on left-hand side) down to hardware components (on the right-hand side). This diagram can therefore be used to better understand how a distributed software system takes advantage of the underlying hardware components to execute a given workload. In particular, one can observe that 91 % of the energy is spent by the CPU package, while the Consumer service drains 65 % of the energy consumption of the monitored scenario. Interestingly, one can observe that this energy consumption is evenly distributed across the 5 slaves, thus fully benefiting from the pod replication support of KUBERNETES. The observed energy overhead is not due to the saturation of a single node, but rather seems to be distributed across the nodes, therefore highlighting an issue in the code of the Consumer service. This issue is related to the acknowledgement of write requests by the CASSANDRA service, which prevents the CONSUMER service to process pending messages.

We believe that, thanks to SMARTWATTS, system administrators and developers can collaborate on identifying energy hotspots in their deployment and adjusting the configuration accordingly.

VI. CONCLUSION

Power consumption is critical concern in modern computing infrastructures, from clusters to data centers. While the state of practice offers tools to monitor the power consumption at a coarse granularity (*e.g.*, nodes, sockets), the literature fails to propose generic power models, which can be used to estimate the power consumption of software artefacts.

In this paper, we therefore reported on a novel approach, named SMARTWATTS, to deliver per-container power estimations for PKG and DRAM components. In particular, we propose to support self-calibrating power models to estimate the PKG and DRAM power consumption of software containers. Unlike static power models that are trained for a specific workload, our power models leverage sequential learning principles to be adjusted online in order to match unexpected workload evolutions and thus maximize the accuracy of power estimations.

While we demonstrate this approach using Intel RAPL and the Linux's *perf_events* interface, we strongly believe that it can be used as a solid basis and generalized to other architectures and system components. In particular, we are working on the validation of our approach with AMD Ryzen architecture (including a support for RAPL).

Thanks to SMARTWATTS, system administrators and developers can monitor the power consumption of individual containers and identify potential optimizations to apply in the distributed system they manage. Instead of addressing performance issues by adding more resources, we believe that SMARTWATTS can favorably contribute to increase the energy efficiency of distributed software systems at large.

ACKNOWLEDGEMENT

The authors would like to thank Joël Penhoat and Pierre Rust from Orange Labs for their insightful feedbacks on this version of the paper.

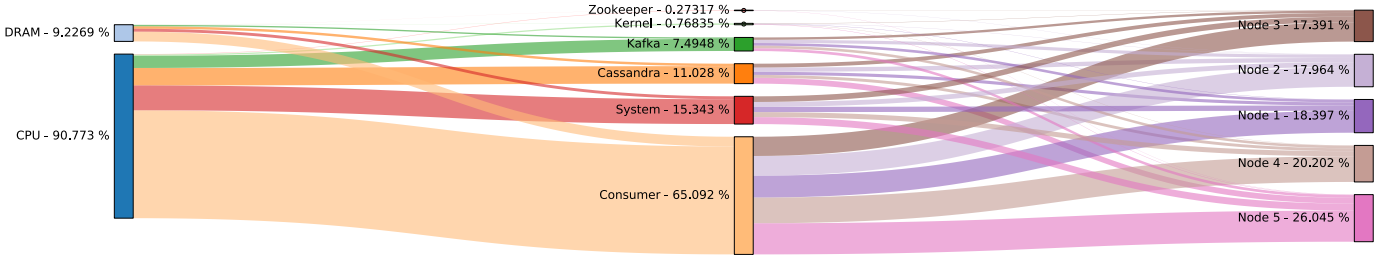


Fig. 8. Distribution of the energy consumption across nodes and resources

REFERENCES

[1] A. Noureddine, R. Rouvoy, and L. Seinturier, "Monitoring energy hotspots in software - Energy profiling of software code," *Autom. Softw. Eng.*, 2015.

[2] D. C. Snowdon, E. L. Sueur, S. M. Petters, and G. Heiser, "Koala: a platform for OS-level power management," in *EuroSys*. ACM, 2009, pp. 289–302.

[3] M. Colmant, R. Rouvoy, M. Kurpicz, A. Sobe, P. Felber, and L. Seinturier, "The Next 700 CPU Power Models," *Journal of Systems and Software*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301377

[4] M. LeBeane, J. H. Ryoo, R. Panda, and L. K. John, "Wattwatcher: Fine-grained power estimation for emerging workloads," in *Computer Architecture and High Performance Computing (SBAC-PAD), 2015 27th International Symposium on*, 2015.

[5] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, "Process-level Power Estimation in VM-based Systems," in *Proceedings of the 10th European Conference on Computer Systems*, 2015.

[6] M. Rashti, G. Sabin, D. Vansickle, and B. Norris, "WattProf: A Flexible Platform for Fine-Grained HPC Power Profiling," in *2015 IEEE International Conference on Cluster Computing*, 2015.

[7] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[8] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, 2012.

[9] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of DRAM RAPL power measurements," in *Proceedings of the Second International Symposium on Memory Systems, MEMSYS 2016, Alexandria, VA, USA, October 3-6, 2016*, B. Jacob, Ed. ACM, 2016, pp. 455–470. [Online]. Available: http://doi.acm.org/10.1145/2989081.2989088

[10] F. Bellosa, "The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems," in *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System*, 2000.

[11] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual Machine Power Metering and Provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010.

[12] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the Effectiveness of Model-based Power Characterization," in *Proceedings of the USENIX Annual Technical Conference*, 2011.

[13] D. Versick, I. Wassmann, and D. Tavangarian, "Power Consumption Estimation of CPU and Peripheral Components in Virtual Machines," *SIGAPP Appl. Comput. Rev.*, 2013.

[14] W. Bircher and L. John, "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems Software*, ser. ISPASS '07, 2007.

[15] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A Comparison of High-level Full-system Power Models," in *Proceedings of the Conference on Power Aware Computing and Systems*, 2008.

[16] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and Responsive Power Models for Multicore Processors Using Performance Counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*, 2010.

[17] Y. Zhai, X. Zhang, S. Eranian, L. Tang, and J. Mars, "HaPPy: Hyperthread-aware Power Profiling Dynamically," in *Proceedings of the USENIX Annual Technical Conference*, 2014.

[18] R. Zamani and A. Afsahi, "A Study of Hardware Performance Monitoring Counter Selection in Power Modeling of Computing Systems," in *Proceedings of the 2012 International Green Computing Conference*, 2012.

[19] M. F. Dolz, J. Kunkel, K. Chasapis, and S. Catalán, "An analytical methodology to derive power models based on hardware and software metrics," *Computer Science - Research and Development*, 2015.

[20] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[21] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunities," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005.

[22] G. Contreras and M. Martonosi, "Power Prediction for Intel XScale® Processors Using Performance Monitoring Unit Events," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005.

[23] T. Li and L. K. John, "Run-time Modeling and Estimation of Operating System Power Consumption," *SIGMETRICS Perform. Eval. Rev.*, 2003.

[24] H. Yang, Q. Zhao, Z. Luan, and D. Qian, "iMeter: An integrated {VM} power model based on performance profiling," *Future Generation Computer Systems*, 2014.

[25] M. Y. Lim, A. Porterfield, and R. Fowler, "SoftPower: Fine-grain Power Estimations Using Performance Counters," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010.

[26] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: An os facility for fine-grained power and energy management on multicore servers," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13. New York, NY, USA: ACM, 2013, pp. 65–76. [Online]. Available: http://doi.acm.org/10.1145/2451116.2451124

[27] E. van der Kouwe, D. Andriesse, H. Bos, C. Giuffrida, and G. Heiser, "Benchmarking Crimes: An Emerging Threat in Systems Security," *CoRR*, vol. abs/1801.02381, 2018.

[28] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber *et al.*, "The NAS parallel benchmarks," *International Journal of High Performance Computing Applications*, 1991.

[29] M. Kurpicz, A. Orgerie, and A. Sobe, "How much does a vm cost? energy-proportional accounting in vm-based environments," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb 2016, pp. 651–658.

[30] G. Prekas, M. Primorac, A. Belay, C. Kozyrakis, and E. Bugnion, "Energy Proportionality and Workload Consolidation for Latency-critical Applications," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 2015.

[31] M. Colmant, P. Felber, R. Rouvoy, and L. Seinturier, "WattsKit: Software-Defined Power Monitoring of Distributed Systems," in *CCGrid*. IEEE Computer Society / ACM, 2017, pp. 514–523.

[32] R. Lupton and J. Allwood, "Hybrid sankey diagrams: Visual analysis of multidimensional data for understanding resource use," *Resources, Conservation and Recycling*, vol. 124, pp. 141 – 151, 2017.