HAL
open science

Process-level Power Estimation in VM-based Systems

Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, Anita Sobe

To cite this version:

Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, et al.. Process-level Power Estimation in VM-based Systems. European Conference on Computer Systems (EuroSys), Apr 2015, Bordeaux, France. pp.14, 10.1145/2741948.2741971 . hal-01130030

HAL Id: hal-01130030

https://inria.hal.science/hal-01130030v1

Submitted on 23 Apr 2015

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.

Process-level Power Estimation in VM-based Systems

Maxime Colmant
ADEME / University of Lille / Inria
maxime.colmant@inria.fr

Mascha Kurpicz
University of Neuchâtel
mascha.kurpicz@unine.ch

Pascal Felber
University of Neuchâtel
pascal.felber@unine.ch

Loïc Huertas
Inria / University of Lille
loic.huertas@inria.fr

Romain Rouvoy
University of Lille / Inria
romain.rouvoy@univ-lille1.fr

Anita Sobe
University of Neuchâtel
anita.sobe@unine.ch

Abstract

Power estimation of software processes provides critical indicators to drive scheduling or power capping heuristics. State-of-the-art solutions can perform coarse-grained power estimation in virtualized environments, typically treating *virtual machines* (VMs) as a black box. Yet, VM-based systems are nowadays commonly used to host multiple applications for cost savings and better use of energy by sharing common resources and assets.

In this paper, we propose a fine-grained monitoring middleware providing real-time and accurate power estimation of software processes running at any level of virtualization in a system. In particular, our solution automatically learns an application-agnostic power model, which can be used to estimate the power consumption of applications.

Our middleware implementation, named BITWATTS, builds on a distributed actor implementation to collect process usage and infer fine-grained power consumption without imposing any hardware investment (*e.g.*, power meters). BITWATTS instances use high-throughput communication channels to spread the power consumption across the VM levels and between machines. Our experiments, based on CPU- and memory-intensive benchmarks running on different hardware setups, demonstrate that BITWATTS scales both in number of monitored processes and virtualization levels. This non-invasive monitoring solution therefore paves the way for scalable energy accounting that takes into account the dynamic nature of virtualized environments.

1. Introduction

Context. Energy-efficient computing is becoming increasingly important. Among the reasons, one can mention the massive consumption of large data centers, estimated to account for about 2% of global greenhouse gas and some of which consume as much as 180,000 homes [8, 28]. This trend, combined with environmental concerns makes energy efficiency a prime technological and societal challenge.

Researchers and operators have been proposing solutions to increase energy efficiency at all levels, from application to runtime and to hardware. As surveyed by Orgérie et al. [22], examples include methods for energy-based task scheduling, energy-efficient software, dynamic frequency and voltage scaling, and energy-aware workload consolidation using virtualization. Virtualization offers environment and performance isolation and, hence, is the basis for many data center and cloud management frameworks. In order to improve their energy efficiency, such cloud management frameworks need to know the resource requirements of the running entities.

For data center providers and users, it is particularly useful to identify which applications are the largest power consumers. However, physical power meters and components with embedded energy sensors are often missing, and they require significant investment and efforts to be deployed a posteriori in a data center. Additionally, these hardware facilities usually only provide system-level or device-level granularity. Hence, software-based power estimation is becoming an economical alternative [22]. Power estimation is relatively accurate when one has full control over the underlying hardware and detailed knowledge of its properties. It typically works by sampling the activity of applications and measuring the power consumption of the whole system using hardware-specific probes.

In virtualized environments, one does not have direct access to the physical CPUs and one can only observe the processor emulated by the *virtual machine*'s (VM) hypervisor. Furthermore, the physical resources available to the emulated CPU may change dynamically as a result of VM scheduling—a VM may run alone on some physical core(s) for some time and later compete with other VMs—or even migrate to another host.

Current approaches providing power estimation remain poorly adapted to virtualized environments and do not provide acceptable measures. The few existing approaches either consider the VM as a black-box running a single applica-

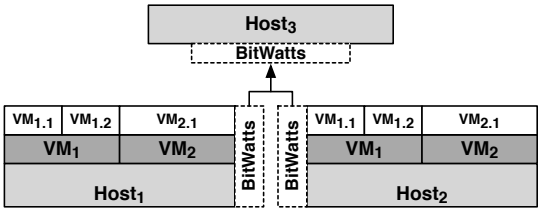

Figure 1. Example for BITWATTS acting in a multi-tenant virtual environment.

tion [11, 14], or they require extensions to the hypervisor or to the host and guest operating systems for being operational [25, 27].

Motivating scenarios. The introduction of fine-grained power monitoring within virtualized environments opens up for new scenarios.

Platform-as-a-service (PaaS) infrastructures such as Google App Engine allow developers to create programs that run in sandbox mode [24]. Request and database handling is performed outside of an application in separate tasks. To isolate which application draws the most power, it is necessary to cover each individual process. This does not only allow for new power-aware pricing models, but also helps improve energy proportionality mechanisms.

In cases of dedicated cloud offers[1] or nested virtualization, such as proposed by Ben-Yehuda et al. [2], an *infrastructure-as-a-service* (IaaS) provider could offer user-controlled hypervisors within a VM. This allows cloud users to run their favorite types of hypervisors and accompanying VMs. However, the management of VMs in such environments can become deeply complex and, with current solutions, it is impossible to monitor the power consumption of a single VM at the highest level of nesting. This prevents typical tasks, such as resource and power provisioning. Such use cases therefore require a flexible solution that can operate on local, nested, and distributed levels without extra efforts.

More specifically, consider a distributed setup with nested virtualization in which we would like to track the power consumption per VM and per user in order to apply power-aware pricing. Such a setup is illustrated in Figure 1. One VM per user can be initially started on each node, and the user can subsequently launch additional VMs running multiple processes within the provided environment. In such settings, it is desirable to be able to monitor the power consumption of each of the user's processes and VMs separately. Furthermore, as a user might operate on multiple nodes, distributed monitoring of the energy consumption of all his processes is also instrumental to determine per-user energy consumption for the pricing model. Our BITWATTS system, which we present in this paper, provides such facilities.

Contributions. In this paper, we propose BITWATTS, a middleware solution to estimate the power consumption of software processes running in virtualized environments.[2] While BITWATTS is a modular framework that can accommodate different power models (including *running average power limit* (RAPL) probes and power meters), we propose a process-level power model, which is application-agnostic and accounts for virtualization—*i.e.*, for emulated cores within a VM—and for the power-aware extensions of modern processors, notably hardware threading and *dynamic voltage and frequency scaling* (DVFS). In our software solution, we expose power probes from the host *operating system* (OS) to the guest OS so that BITWATTS can estimate the power consumption of processes running within a VM. In addition, this design can operate in distributed settings, with consumption information transmitted over high-performance publish/subscribe middleware.

We have implemented BITWATTS in Scala, as an extension of the POWERAPI actor toolkit[3]. We ran extensive experiments with several workloads on various computer settings, and we assessed the accuracy of BITWATTS by comparing its output to physical measurements performed with a power meter. Results indicate that BITWATTS provides trustworthy power estimation within a few per cent of actual measures when configured with the appropriate power model for the underlying hardware. We describe the design of such a CPU power model, which is application-agnostic, supports both CPU- and memory-intensive workloads and is processor-aware, including *multi-cores*, *hyper-threading*, *dynamic scaling*, and *dynamic overclocking* features.

In a typical server, the major power consumer is the CPU [22], covering at least one third of the overall power consumption. Hence, like other studies [3, 10, 18, 32], our power model focuses on processor consumption and accurately monitors applications that are CPU- and memory-intensive. For disk-intensive workloads, we need further studies and finer-grained models since two hard disks, even of the same model and making, have different power consumption patterns [15]. Therefore, we selected our benchmarks in such a way that the additional power possibly consumed by the disk is negligible. Studies in data centers [22] showed that network I/O (in the case of Ethernet) is not impacting the power consumption as the difference between idle and fully utilized links is negligible.

Outline. The rest of the paper is organized as follows. We first discuss related work in Section 2. We then introduce the general principle and the architecture of BITWATTS in Section 3 and describe the power models in Section 4. We provide an in-depth evaluation in Section 5 and finally, conclude in Section 6.

[1] `https://www.ovh.com/ca/en/dedicated-cloud`

[2] Available as open source at: `http://bitwatts.powerapi.org`

[3] POWERAPI (AGPL): `http://powerapi.org`

2. Related Work

Models for power estimation have been mainly studied at the level of processors, and less extensively in the context of virtualization. We give an overview of previous research on both aspects in the rest of this section.

2.1 CPU Power Models

As current platforms do not provide fine-grained power measurement capabilities, McCullough *et al.* [19] argue that power models are the first step to enabling dynamic power management for power proportionality on all levels of a system. Currently, the approach closest to hardware-based monitoring is the *running average power limit* (RAPL) feature available for the Intel Sandy Bridge and Ivy Bridge CPUs [9], which allows for monitoring the power consumption of the whole CPU package.

As this feature is not available on other CPUs, power models typically rely on a number of performance counters. For example Li and John [17] use 5 counters, including the *instructions per cycle* (IPC) counter, and rely on a regression model for estimation. Similar work has been performed by Contreras et al. [7] who additionally consider different CPU frequencies, but not multi-core architectures. Bircher and John [5] managed to reduce the error of power models to 9 % using performance counters for component-level power estimation. Other work starts with all available counters and then try to reduce their number [23] by analyzing the correlation between counters of different architectures and power dissipation. Usually the accuracy of the models is validated by comparing estimates with the measures of a power meter when running benchmarks in isolation [31].

Power modeling often considers learning techniques—for example based on sampling [3]—that assume the proportionality of system events to power consumption. Measurements of a hardware power meter are gathered and subsequently used, together with a set of normalized estimated values, in various regression models, which are so far mostly linear [19].

Kansal *et al.* [11] and Versick *et al.* [30] notably point out that linear power models depending on the CPU load are not sufficient anymore and that more parameters have to be considered. McCullough *et al.* [19] show that, especially in multi-core systems, linear models lead to a much higher mean relative error of 10-14 % for CPU power consumption and cannot easily be improved by applying more complex techniques. Linear models rely on the independence of the covered features, which is not realistic in current systems. Polynomial/exponential regression can cover these dependencies and, as shown in [5], a quadratic solution better fits the power modeling of multi-core systems. The described systems must however isolate processor features, such as *HyperThreading* and *TurboBoost*, to avoid hidden states. HAPPY [32] introduces a hyperthread-aware power model that differentiates between the cases where either single or both hardware threads of a core are in use.

Shen *et al.* [25] propose power containers to manage energy and power usage on multi-core servers and cloud computing platforms. The authors evaluate power containers with multiple applications, but each of them is considered separately. In the evaluation of BITWATTS, we also cover scenarios with multiple applications running in parallel. Power containers are also tied to physical hosts and not evaluated in a virtual environment.

2.2 VM Power Models

In data centers, the efficiency of VM consolidation, power dependent cost modeling, and power provisioning are highly dependent on accurate power models [29]. Such models are particularly needed because it is not possible to attach a power meter to a virtual machine [16]. In general, VMs can be monitored as black-box systems for coarse-grained scheduling decisions. If we want to be able to do fine-grained scheduling decisions—*e.g.*, with heterogeneous hardware—we need to be able to consider finer-grained estimation at sub-system level and might even need to step inside the VM.

So far, fine-grained power estimation of VMs required profiling each application separately. One example is WATTAPP [14], which relies on application throughput instead of performance counters as a basis for the power model. The developers of PMAPPER [29] argue that application power estimation is not feasible and instead perform resource mapping using a centralized step-wise decision algorithm.

To generalize power estimation, some systems like JOULEMETER [11] assume that each VM only hosts a single application and thus treat VMs as black boxes. In a multi-VM system, they try to compute the resource usage of each VM in isolation and feed the resulting values in a power model.

Bertran *et al.* [3] propose an approach closer to BITWATTS. They use a sampling phase to gather data related to *performance-monitoring counters* (PMCs) and compute energy models from these samples. With the gathered energy models, it is possible to predict the power consumption of a process, and therefore apply it to estimate the power consumption of the entire VM.

Another example is given by Bohra *et al.* in [6], where the authors propose a tool named VMETER that estimates the consumption of all active VMs on a system. A linear model is used to compute the VMs' power consumption with the help of available statistics (processor utilization and I/O accesses) from each physical node. The total power consumption is subsequently computed by summing the VMs' consumption with the power consumed by the infrastructure.

Janacek *et al.* [10] use a linear power model to compute the server consumption with *postmortem* analysis. The computed power consumption is then mapped to VMs depending on their load. This technique is not effective when runtime information is required.

In Stoess *et al.* [27] the authors argue that, in virtualized environments, energy monitoring has to be integrated within the VM as well as the hypervisor. They assume that each

device driver is able to expose the power consumption of the corresponding device as well as an energy-aware guest operating system and is limited to integer applications.

2.3 Synthesis

As a summary of the current state of practice, the existing CPU power models found in the literature cannot be reproduced because *i)* the details of the selected counters are not provided [31] or sufficiently documented [32], *ii)* they are tailored to a specific processor architecture (including a limited set of power-aware features) [18], or *iii)* they build on private workloads that cannot be reused to assess alternative power models [32]. BITWATTS differs from the state-of-the-art by providing an open source implementation of the proposed toolkit and builds on standard counters and benchmarks (*e.g.*, `stress`, PARSEC, SPECJBB) to provide an open testbed for CPU power models.

More specifically speaking, one could note that the literature has been mostly focusing on the definition of power models for physical machines by trying to cover some of the power-aware features of multi-core processors. Nevertheless, the existing approaches consider each feature separately and, to the best of our knowledge, none of them provide a CPU power model that accounts for all of these features in the context of multi-core systems that run several applications concurrently.

With regard to the power consumption of VMs, state-of-the-art solutions provide no or limited support for fine-grained monitoring of applications running within a VM. The few existing approaches either consider the VM as a black-box running a single application, or they require extensions to the hypervisor or to the host and guest operating systems for being operational.

In this paper, we therefore propose to tackle both challenges by reporting on the design of software-defined power meters that can run both on the host and in a VM. In particular, on the host, we propose a first configuration of a software-defined power meter that builds on a new CPU power model that accounts for common power-aware features of multi-core processors to deliver accurate power estimations at the granularity of a software process. In the VM, we introduce a second configuration of a software-defined power meter that connects to the host configuration in order to distribute the power consumption of VM instances between the hosted applications. The proposed configuration can even be extended to consider distributed power monitoring scenarios involving application components spread across several host machines.

Unlike existing approaches found in the literature, the CPU power models we describe *i)* are application-agnostic, *ii)* are processor-aware, and *iii)* scale with the number of software processes to be monitored concurrently. We assess these properties by reporting on the errors observed for both CPU-intensive and memory-intensive applications provided by acknowledged benchmarks.

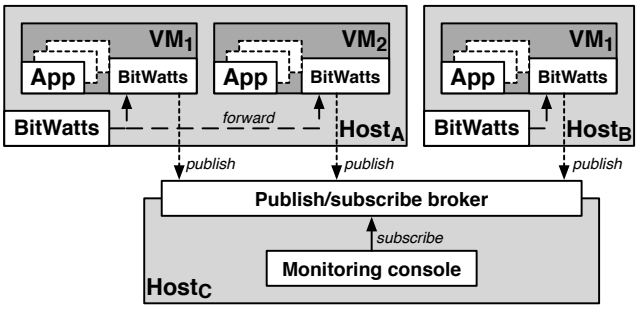

Figure 2. BITWATTS middleware architecture.

3. Software-defined Power Meters

Power estimation of processes running in virtualized environments is not a trivial task, since several factors have to be considered. In particular assumption, such as the presence of a single application running in a single VM on a single core, do not hold anymore. One has to deal with complex scenarios with a number of VMs that may exceed the number of physical cores and several applications that run within each VM. To cope with these different dimensions of scaling, we designed and implemented the BITWATTS middleware framework as a modular solution to build software-defined power meters. In the rest of this section, we give a high-level overview of its architecture and implementation.

3.1 Architecture Overview

BITWATTS relies on a multi-tier architecture, depicted in Figure 2, that shares the power consumption of the VMs running on the host to the application processes running within the VM. Since the VM does not have direct access to the hardware, we use a fast communication interface to connect instances of BITWATTS running on the host and in the VMs. Similarly, BITWATTS also supports communication across machines using publish/subscribe communication channels to report consolidated power estimations of distributed applications spanning multiple nodes (*e.g.*, in a cluster).

3.2 Power Meter Middleware Toolkit

We built BITWATTS as a modular middleware solution to assemble software-defined power meters. Software-defined power meters are customizable solutions that can deliver power consumption reports at various frequencies and granularity, depending on the power monitoring requirements. In particular, this paper focuses on per-second process-level monitoring in order to closely monitor the activity of an application running on a system.

Our solution builds on the POWERAPI toolkit [20], which adopts the actor programming model as a solution that can scale with the frequency and the number of applications to be monitored. The software components of BITWATTS are therefore implemented as actors, which can process millions of messages per second, a key property for supporting real-time power estimation. More specifically, the POWERAPI

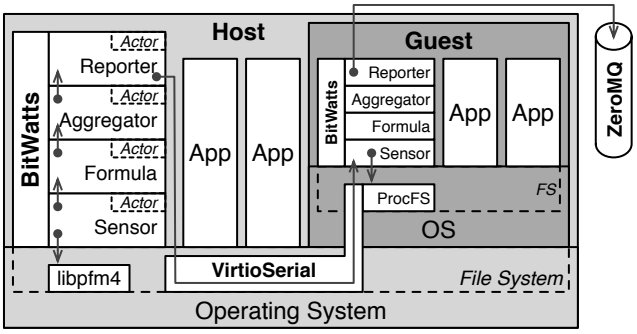

Figure 3. BITWATTS middleware implementation.

toolkit identifies four types of actors that are reused and extended in BITWATTS:

Sensor connects the software-defined power meters to the underlying system in order to collect raw measurements of system activity. Raw measurements can be coarse-grained power consumption reported by third-party power meters and embedded probes, or CPU activity statistics as delivered by the process file system (`ProcFS`). Sensors are triggered according to the requested monitoring frequency and forward raw measurements to the appropriate formula.

Formula uses the raw measurements received from the sensor to compute a power estimation. A formula therefore implements a specific power model [11, 14, 30] to convert raw measurements into power consumption. The granularity of the power consumption reported by the formula (machine, core, process) depends on the granularity of the measurements forwarded by the sensors.

Aggregator is in charge of aggregating power consumption, according to a specific dimension like the *process identifier*, to compute the energy consumption, or *timestamp*, to group the power consumption of several applications.

Reporter finally formats the power consumption produced by the formula or the aggregator into a suitable format. Such reports can be provided for instance via a Web interfaces or a virtual file system (*e.g.*, based on FUSE).

As the BITWATTS middleware framework supports process estimation in VM-based systems, implementations of the sensor, formula, and reporter actors are assembled in different configurations on the hosts as well as in the VMs (see Figure 3).

Additionally, to improve the accuracy of state-of-the-art power estimation, we deliver a new power model that builds upon a libpfm[4] sensor actor on the host to collect the hardware performance counters associated to the monitored VM process. The formula actors consume the measurements collected on the host by this libpfm sensor to estimate the power consumption of a given process (see Section 4.1). The resulting consumption measures are automatically published by two reporter actors through two different communication channels: VirtioSerial[5] and in a distributed setup also to ZeroMQ.[6] The data forwarded through these channels is consumed by sensor actors.

The BITWATTS middleware framework therefore provides an exhaustive toolkit to assemble software-defined power meters on demand. The results reported in the following sections are notably based on a variety of software-defined power meters built with BITWATTS to: monitor coarse-grained power consumption using a third-party power meter or RAPL probes, learn the power model of the processor, deliver process-level power consumption on the host, and report on fine-grained power consumption within the VMs.

3.3 Power Consumption Communication Channels

Exchanging data between instances of BITWATTS requires two levels of communication. First, we need to exchange data between the host and the VM to estimate the power consumption of a process within the VM. Second, in a distributed setup, we want BITWATTS to report the power estimation to another server, *e.g.*, to aggregate the data monitored on multiple physical or virtual nodes.

For the hierarchical communication between instances of BITWATTS running on the host and a VM, a lightweight transport mechanism is required to exchange messages at a high rate while crossing the VM boundaries.

VirtioSerial is based on the file system and has been developed for the very purpose of inter-VM communication. It provides the performance required to reduce likelihood of synchronization errors of power measurements between host and virtual machine.

The VirtioSerial hierarchical communication channel is implemented in BITWATTS as a reporter actor on the host and a sensor actor in the VM (see Figure 3). Multiple instances of BITWATTS are running concurrently: one in the host and one per VM. For the host, the VirtioSerial reporter communicates the power consumption of the VM process to the `virtio-pci` device. In the VM, the VirtioSerial sensor connects to the VirtioSerial port and reads power consumption reported by the host. The BITWATTS formula uses these values to compute the process-level power consumption within the VM.

In a distributed setup, we need to communicate across machines, typically to aggregate the power measurements from distributed application components running on different VMs and hosts. Our distributed communication channel therefore consists of a publish/subscribe system using ZeroMQ. ZeroMQ is a networking API that supports complex messaging patterns and provides bindings for various programming languages while being lightweight. The key component of the publish/subscribe system is the broker. It forwards messages received from the distributed BITWATTS instances to interested subscribers, for example loggers or the monitoring

[4] `http://perfmon2.sourceforge.net`

[5] `http://www.fedoraproject.org/wiki/Features/VirtioSerial`

[6] `http://www.zeromq.org`

console (see Figure 2). Messages exchanged between BITWATTS, the broker, and the subscribers are serialized using Apache Thrift,[7] an efficient interface definition language and binary serialization protocol.

4. Process-level Power Models

BITWATTS relies on specific power models to estimate power consumption of individual processes. Per-process power estimation is a cornerstone to identifying the largest power consumers and to take informed decisions. In particular, we discuss in this section how these models support multi-core architectures including power-aware features as well as how such power models can be connected to support power estimations within a VM.

4.1 Multi-core CPU Power Model

Power-aware processors. To control energy consumption, CPUs rely on frequency scaling and power saving modes to adjust their performance according to computation requirements. In particular, the multi-core processors designed by Intel integrate the following features:

Hyper-Threading (HT) is used on some processor generations (*e.g.*, Pentium IV, Xeon) to separate each core into two threads. The technology is based on the *simultaneous multi-threading* (SMT) principle, which allows the processor to seamlessly support *thread-level parallelism* (TLP) in hardware and share more effectively the available resources. Performance gains strongly depend on software parallelism, and for a single-threaded application it may be more effective to actually disable this technology.

SpeedStep (SS) is Intel's implementation of *dynamic voltage/frequency scaling* (DVFS), which allows a processor to adjust its clock speed and run at different frequencies or voltages upon need. The OS can increase the frequency to quickly execute operations or reduce it to minimize dissipated power when the processor is under-utilized.

TurboBoost (TB) can dynamically increase the processor frequency beyond the maximum bound, which can be greater than the *thermal design power* (TDP), for a limited period of time. It therefore allows the processor cores to execute more instructions by running faster. TurboBoost is however only activated when some specific conditions are met, notably related to the number of active cores and the current CPU temperature. It also depends on the OS, which may request to trigger it when some applications require additional performance.

[7] http://thrift.apache.org

Vendor	Intel	Intel
Processor	i3	Xeon
Model	2120	W3520
Design	4 threads	8 threads
Frequency	3.10 GHz	2.66 GHz
TDP	65 W	130 W
SS	✓	✓
HT	✓	✓
TB	✗	✓

Table 1. Intel processor specifications.

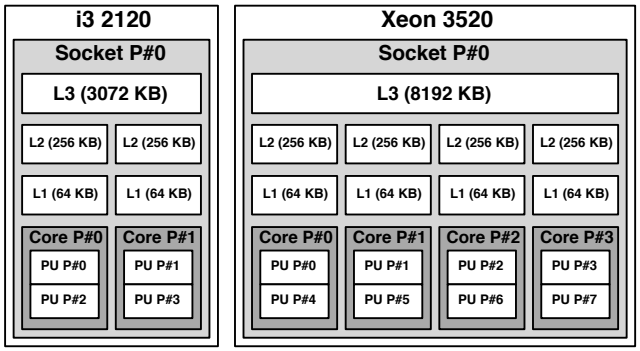

Figure 4. Core i3 and Xeon topologies.

As an illustration, Table 1 reports on the configuration of two families of Intel processors that exhibit different features and are used in our evaluation of BITWATTS. Their internal complexities are reported by the *portable hardware locality* (hwloc)[8] software package and detailed in Figure 4. These two configurations differ by the number of cores and threads available as well as the CPU features (TurboBoost) that can be exploited by the operating system.

Power model learning. Learning the power model of multi-core processors requires the definition of a workload that carefully stresses the various features it supports. Thereby, it is important to isolate the noise induced by other hardware components to properly capture the consumption of the CPU under evaluation. We therefore choose the `stress`[9] utility, which is available on most of UNIX systems, to perform specific workload scenarios. It allows us to incrementally stress the different hardware components, such as the CPU, the memory, and the disk.

Using the options provided by the `stress` utility, we generate different workloads. First, we stress the processor core by core under full load in order to capture its maximum frequency and to observe the effect of the Hyper-Threading feature on the the power consumption. Then, we dynamically change the CPU load to characterize the effect of the SpeedStep feature on the power consumption. This workload is applied for each frequency made available by the processor, using `cpufreq-utils`. Finally, by stressing an increasing number of cores, we are able to identify the frequencies used by the TurboBoost feature and and the associated power models.

[8] http://www.open-mpi.org/projects/hwloc

[9] http://linux.die.net/man/1/stress

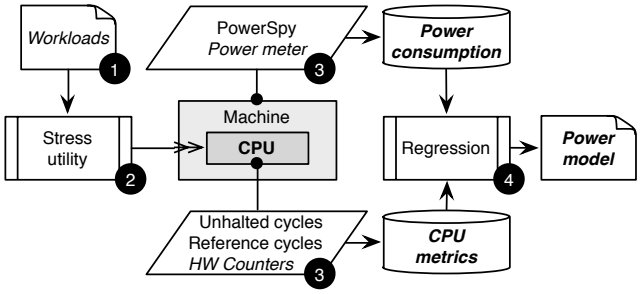

Figure 5. Power model learning process.

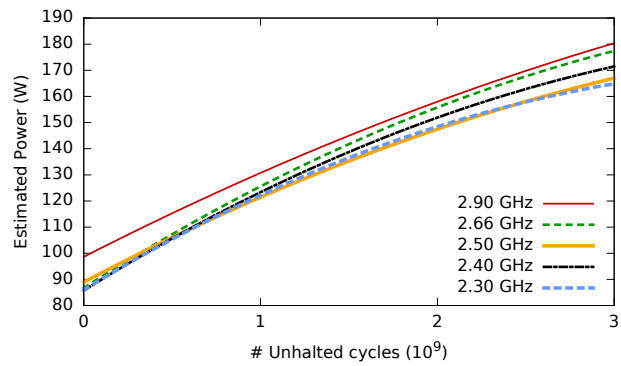

Figure 6. Power models for the highest frequencies on the Xeon processor.

To learn the power model, we then need to collect runtime measurements that faithfully capture the specificity of a large set of CPU workloads. As reported by the authors of [11], the CPU load does not reflect the variety of the processor's activities. We therefore decided to base our model on hardware performance counters to collect low level and accurate metrics reflecting the types of operations that are truly executed by the processor. Specifically, we use the `libpfm4` library for accessing hardware performance counters available on modern processor architectures, regardless of the OS. The hardware performance counters used to estimate the power consumption of processors have to be carefully selected according to two criteria: their availability on a large family of architectures and the overhead imposed by their exploitation.

Our objective is to build a lightweight model that imposes very limited overhead to our middleware solution. We therefore chose as in [18, 31, 32] the `unhalted-cycles` (uc)[10] and `reference-cycles` (rc)[11] counters to accurately characterize the power model of multi-core architectures. While the first counter represents the number of cycles executed and thus the activity of the cores, the second one represents the number of cycles counted at a reference frequency that might differ from the actual speed of the processor; it is therefore very useful to approximate the core frequency, even when the processor triggers the turbo mode.

The average frequency (f) is computed by dividing the number of `unhalted-cycles` by the number of `reference-cycles` ($f = uc/rc$). The average frequency f is used to build the power models and to choose at runtime which counter to apply.

To monitor the power consumption during the learning phase, we consider a power meter that reports on the consumption of the whole machine as "ground truth". Specifically, we used the PowerSpy[12] Bluetooth power meter. Depending on the country, the PowerSpy power meter samples the power consumption between 45 and 65 Hz. As part of this paper, we normalize this frequency by requesting a monitoring window of 250 ms (4 Hz), which is computed as the average consumption monitored by the PowerSpy. To improve the accuracy of the power model, we run the identified workloads several times to reduce the variance introduced by the physical measures.

Power model inference. The hardware performance counters and power information collected during the execution of the workloads are then correlated using a polynomial regression to connect the evolution of the power consumption with the variation of the number of `unhalted-cycles`. We build a model for different processor frequencies that represents the power consumption for a single core, covering the HT feature [32], and we assume that the power consumption grows linearly with the number of active cores. Figure 5 depicts this process, applied on the processor configuration listed in Table 1.

In practice, the power model we obtained for a core on an Intel Xeon processor ($host$) running at a given frequency (f) for a short period of time can be represented by the equation

$$P_{host}(f) = P_{idle}(f) + \sum_{pid \in PIDs} P_{cpu}(f, uc^1_{pid} ... uc^N_{pid})$$

where $P_{idle}(f)$ corresponds to the static power consumption (*i.e.*, the idle power consumption) of the machine for the frequency f that we inferred from the regression step, and $uc^1_{pid} ... uc^N_{pid}$ is a vector of `unhalted-cycles` collected at runtime per active process identifier pid and per core $1..N$. The power consumption of the CPU, P_{cpu}, is defined as the sum of the power consumption per frequency, P_f, for each core n:

$$P_{cpu}(f, uc^1_{pid} ... uc^N_{pid}) = \sum_{n=1}^{N} P_f(uc^n_{pid}).$$

We finally obtain a power model per frequency, including TB-specific frequencies. One of the resulting formulae is described below for the TB frequency of a Xeon processor

[10] `CPU-CLK-UNHALTED:THREAD`, event=0x003c

[11] `CPU-CLK-UNHALTED:REF`, event=0x013c

[12] `http://www.alciom.com/en/products/powerspy2.html`

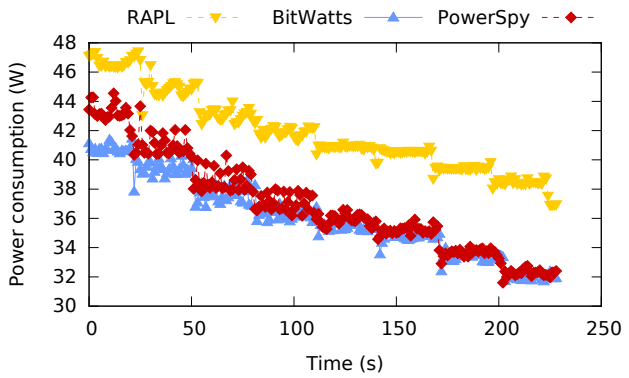

Figure 7. Decreasing load of stress on $i3$ in the host, compared to RAPL.

(2.90 GHz):

$$P_{2.90}(uc_{pid}) = \frac{8.64 \cdot uc_{pid}}{10^9} - \frac{6.10 \cdot uc_{pid}^2}{10^{18}}.$$

The resulting formula is a polynomial of degree 2 (depicted in Figure 6), which is conform to results published in the literature and the impact of the HT feature on the power models [32]. Figure 6 plots the power estimation according to the number of `unhalted-cycles` for each of the power models we inferred per frequency. For the sake of clarity, only the frequencies above 2.30 GHz are reported in the figure. The idle consumption ($P_{idle}(f)$ when $x = 0$) is computed by the regression and is impacted by the current frequency of the processor. One can observe that the 2.50 GHz line is above the 2.40 GHz line, which is mainly due to the inaccuracy of `cpufreq-utils`: it keeps track of the average frequency and might not report exact values at any given time, notably ignoring the turbo frequencies.

Power model assessment. First, to demonstrate that BITWATTS is able to handle applications with diverse load, we start with a baseline experiment on the $i3$. We run the `stress` tool on a single core in combination with `cpulimit`. Every 30 seconds, the stress load is decreased by 10 %. In this experiment, we compare the results not only to PowerSpy, but also to *running average power limit* (RAPL) counters, which report CPU-package power consumption and are available on recent Intel processors (since the *Sandy Bridge* processor generations and hence on the $i3$). Furthermore, for this experiment, we set the CPU frequency to a fixed ratio of 1.6 GHz to avoid peaks in the power measurements of PowerSpy.

Figure 7 shows the results of the workload executed on the host. We see that the RAPL counters follow the trend of the workload, but tend to overestimate the power consumption of a single CPU. Compared to RAPL, BITWATTS provides power estimation that is much closer to PowerSpy that we consider as the ground truth. This indicates that BITWATTS performs accurate sub-system estimation in various load scenarios, which is a prerequisite to be able to monitor virtual machines using a subset of the resources of a physical host.

In the next scenario, we assess our power model for multi-threaded applications in comparison to PowerSpy. This comparison uses the well-known PARSEC [4] v2.1 benchmark suite, which includes many CPU-intensive workloads. This suite was designed to stress all the resources available on multi-core architectures. In particular, we report the power consumption of all the benchmarks available on two different configurations used in our tests. Figures 8 and 9 report the relative error between the measured and estimated power consumption (by aggregating the power consumption per process using P_{host}).

Even though PARSEC was not included as a workload during the sampling phase, one can observe that the estimation produced by our power models is close to the power measurements collected for the two different processor models considered. The closest method to ours, described in [30], adopts an iterative approach to minimize the error rate to at

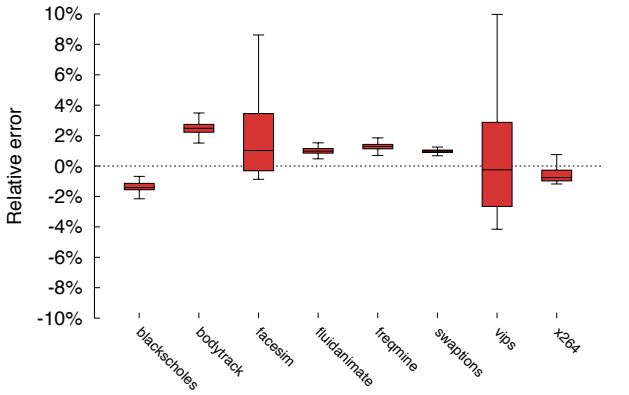

Figure 8. Relative error distribution of the PARSEC benchmarks on the Xeon processor.

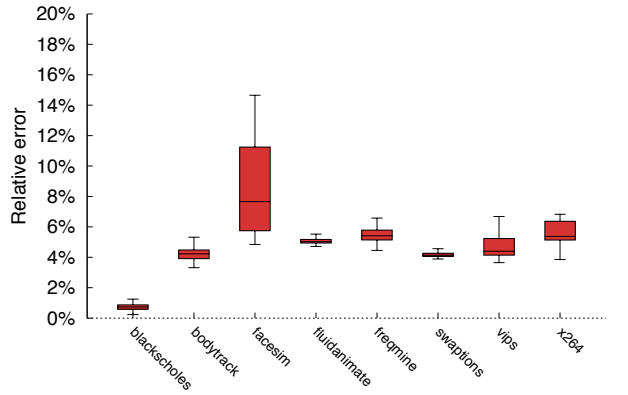

Figure 9. Relative error distribution of the PARSEC benchmarks on the i3 processor.

most 5 %. However, the key limitations of their approach are *i)* they only consider full usage of the cores, and *ii)* they rely on an application-specific model. Our solution is application-agnostic, supporting both CPU- and memory-intensive workloads, and are processor-aware, considering different models of CPUs including *multi-cores*, *hyper-threading*, *dynamic voltage/frequency scaling*, and *dynamic overclocking* features.

Figure 10 illustrates the capability of estimating and isolating the power consumption of concurrent processes running on the same CPU. In particular, it shows how the power consumption of the Intel Xeon configuration is distributed between the idle power consumption and two benchmarks taken from the PARSEC suite (x264 and freqmine). Compared to physical measurements, when running at a frequency of 4 Hz (every 250 ms), our solution achieves a median error of 0.30 % with a maximal error of 9.73 %, thus competing with *post-mortem* analysis like [10].

Regarding the monitoring frequency, BITWATTS is mostly limited by the frequency of the hardware and software sensors used to collect runtime metrics. In particular, BITWATTS can report on the power consumption of software processes up to 40 Hz when connected to the PowerSpy, and up to 10 Hz when using the `libpfm4` library. However, by increasing the monitoring frequency one can observe that the stability of power consumption is affected, which does not help to properly identify the power consumption of the processes. Therefore, in the rest of the paper, we configure BITWATTS to report on the power consumption with a frequency of 1 Hz in order to smooth the reported values.

Additionally, Figure 10 reports on the power consumption of BITWATTS during execution. The power consumption of $5.4\ W$ on average demonstrates that our implementation of the power model has a reasonable footprint and is weakly impacted by the number of processes being monitored. This footprint acknowledges the design and the implementation of BITWATTS as a scalable actor toolkit to build software-defined power meters.

Generality of the model. While the multi-core CPU power model proposed in this paper is only assessed on Intel processors (see Table 1), the solution that we describe does not rely on any Intel-specific extensions. Indeed, our model considers processor features (HT, SS, TB) that are also available from other vendors. In particular, AMD processors also represent a target CPU architecture for our power model, but a limitation of the `libpfm4` library currently prevents BITWATTS to access the `reference-cycles` to compute the current frequency. Once this barrier is lifted, we expect to be able to also demonstrate the validity of our model on AMD processors with results similar to those reported for Intel.

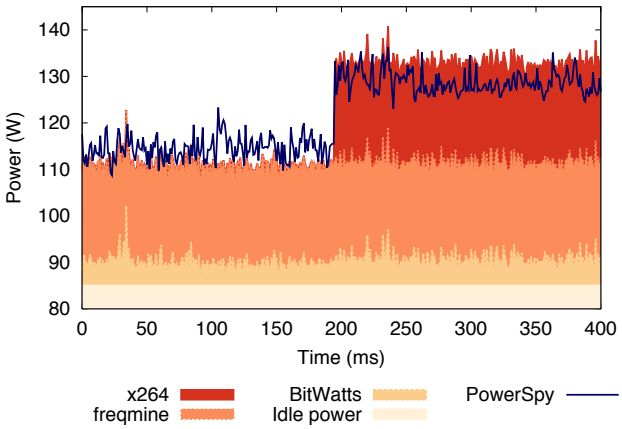

Figure 10. Process-level power consumption of BITWATTS, x264, and freqmine on the Xeon processor.

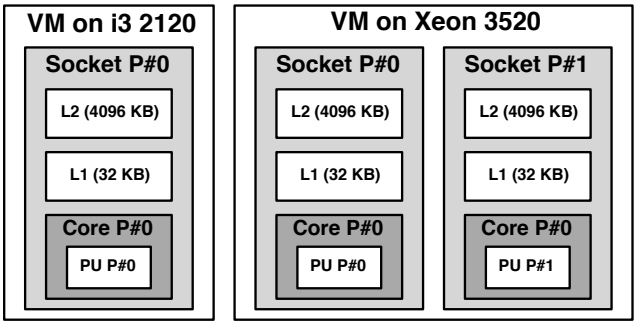

Figure 11. Core i3 and Xeon VM topologies.

4.2 Virtual CPU Power Model

Unlike the architectures observed at the host level (see Figure 4), virtual CPUs tend to be simpler: they map physical cores to logical processors (sockets) and typically do not support any SS/HT/TB features, as illustrated in Figure 11. Hence, when pinning a single-core VM on a physical core of the host, the power consumption of a process running in the VM is proportional to the CPU utilization of the VM on the host.

To estimate the power consumption of an application running in the VM $P_{vm}(app)$, we need therefore to know the consumption of the VM process $P_{cpu}(vm)$ on the host machine, as well as the CPU utilization of the application $U_{vm}(app)$ relatively to the other applications running in the VM $U_{vm}(total)$:

$$P_{vm}(app) = P_{cpu}(f, uc^1_{vm} ... uc^N_{vm}) \cdot \frac{U_{vm}(app)}{U_{vm}(total)}.$$

BITWATTS uses a sensor in the VM to monitor the utilization of the application under observation and of all processes running in the VM. Another sensor gathers information about

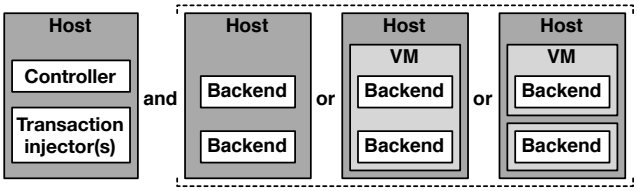

Figure 12. Possible setup of SPECJBB (only backends are part of the evaluation).

the power consumption of the VM forwarded by the host. The formula then computes the power consumption based on the model and forwards the results to a reporter. Note that this reporter can be used to implement distributed energy monitoring scenarios using publish/subscribe middleware, as described in the next section.

5. Evaluation of BITWATTS

In this section, we report on the experimental results we obtained for BITWATTS. In particular, we show that accurate host power estimation and efficient communication with the VM are necessary to support power estimation in realistic virtualized environments.

5.1 Experimental setup

The experimental setup consists of two types of servers (i3 and Xeon) with different hardware characteristics, as shown in Table 1. For the distributed setups, we use three identical servers of type i3.

We rely on KVM [12] for virtualization. KVM turns the Linux kernel into a hypervisor without need for any additional software. In addition to the typical process operating modes (kernel space, user space) of Linux, KVM adds a *guest mode* for programs running in a virtualized environment. This feature helps for measuring the CPU time used by a virtual process.

As KVM does not perform any emulation to run operating systems on various architectures, we combine it with QEMU[13] to emulate different CPU and device types. With QEMU/KVM, the VM runs as a normal user process and is hence controlled by the Linux scheduler. By default, the scheduler tries to keep a process on the same CPUs, notably to maximize cache efficiency. We run KVM/QEMU with an off-the-shelf Ubuntu 13.11 on both server types (i3 and Xeon).

We want to investigate in our experiments the accuracy and applicability of BITWATTS at different scales. Therefore, we first consider the execution of benchmarks on a single host, with an increasing number of concurrently running VMs, to observe the impact of VM scheduling on the host. As a first benchmark, we use PARSEC [4] v2.1 for our experiments, as it is multi-threaded and CPU-intensive. PARSEC contains a variety of applications implemented in C. We experiment

[13] `http://www.qemu.org`

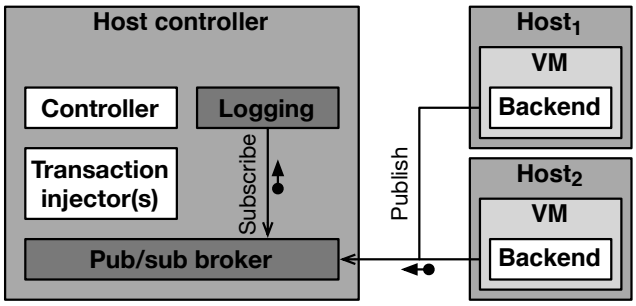

Figure 13. Distributed SPECJBB setup.

with all except two (raytrace, ferret) that were not readily supported by our hosts. We use the PARSEC *native* workload as it yields sufficiently long execution times. We allocate 2 threads per VM, thus allowing the execution of 4 concurrent VMs on the Xeon.

Then, to further evaluate BITWATTS in a real-world, multi-threaded and distributed environment, we use the SPECJBB2013 benchmark [1]. This benchmark implements a supermarket company that handles distributed warehouses, online purchases, as well as high level management operations (data mining). The benchmark is implemented in Java and consists of *controller* components for managing the application and *backends* that perform the actual work. In our experiments, we focus on evaluating the power consumption of the backends, since they can be scaled arbitrarily in virtualized environments. A run takes approximately 45 minutes; it has varying CPU utilization levels and requires at least 2GB memory per backend to finish properly.

In order to have more than one backend run on our instances of i3, we apply the following parameter changes to `specjbb2013.conf`: we reduce the number of customers and products to $50,000$, increase the step-size, and reduce the maximum and minimum duration for phase 2 of the benchmark.[14]

Since we only have several identical servers of type i3, the SPECJBB experiments are only executed on these machines. We compare different setups, running one or two backends on the host or in a VM (Figure 12). The distributed setup consists of a controller host and two virtualized or non-virtualized backend hosts (Figure 13). Note that in virtualized scenarios one BITWATTS instance runs on the host and one in the VM.

5.2 Scaling the Number of VMs

We already assessed the multi-core CPU power model on the host machine, introduced in Section 4.1, by comparing the BITWATTS estimation of PARSEC to the values reported by the PowerSpy. In this section, we first evaluated the virtual CPU power model, described in Section 4.2, by comparing the BITWATTS estimation of PARSEC running in the VM to the values reported by the PowerSpy on the host. In this

[14] Note that these changes make our runs non-compliant, therefore we do not use the SPEC-specific metrics in this paper.

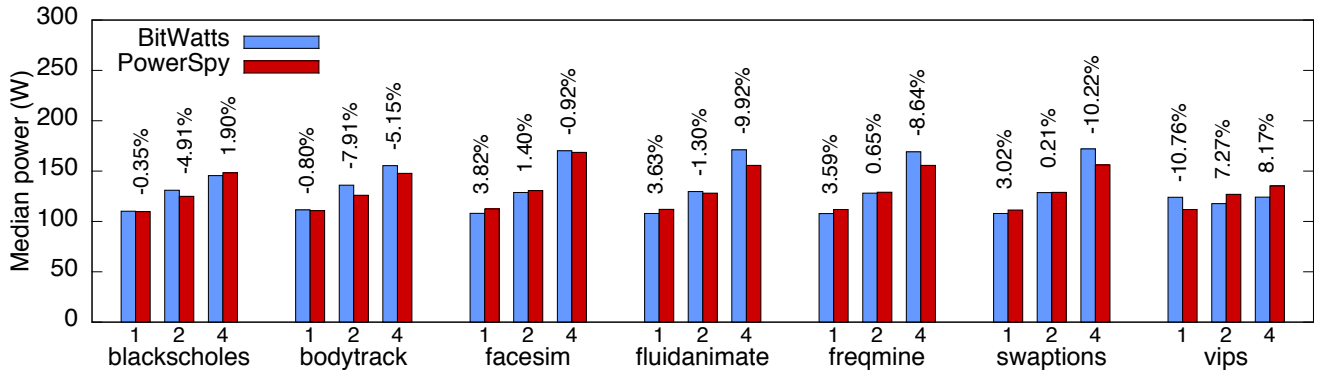

Figure 14. Power consumption of the host when scaling PARSEC on multiple VMs.

experiment, PARSEC is running in a single VM, which has been allocated 2 cores on the host. As the activity of the other active processes is comparably negligible, we compute the BITWATTS estimation as the sum of the power estimation in the VM with the idle consumption ($P_{idle}(f)$) of the host machine. Figure 14 therefore reports on the median power error observed between BITWATTS and PowerSpy. The overall PARSEC experiment resulted in roughly 10,000 power values with a runtime of 1 hour per VM experiment. Note that we did not pin the VM to any specific cores on the host, hence we rely on the native KVM scheduling. When running a single VM on the host, power estimation within the VM by BITWATTS has similar precision to that on the host (see Figure 8). This measure assesses that the multi-core CPU power model we propose properly captures the *guest mode* used by KVM to execute the VMs on the host.

Then, given that nowadays VM-based systems tend to be consolidated to minimize the number of active hosts (*e.g.*, [13]), we evaluate the precision of our software-defined power meter when scaling the number of VMs to be executed on the host. For each of the PARSEC benchmarks, we evaluate the median power error when scaling the number of VMs form 1 to 4 on the Xeon processor. As we do not try to measure the side effects of host over-provisioning on power, we do not exceed the number of physical cores available on the host.

The relative error reported in Figure 14 spans from less than 1 % (fluidanimate) up to around 10 % (swaptions) with increasing errors if the cores used by the VMs reach the number of physical cores on the host. This reflects results found in literature, but in comparison to existing solutions like VMeter[6] we are not only able to report power per VM if multiple VMs run on a single host, but also per process within each VM. This experiment demonstrates that the virtual CPU power model we introduced in Section 4.2 holds in virtual environments, given the simplified architecture of the virtual processor exposed by the hypervisor (see Figure 11).

5.3 Scaling the Number of Hosts

In this section, we evaluate the power consumption of a real-world application (SPECjbb) using BITWATTS. In particular, we further show the possibility of estimating workloads on several nodes such as commonly used in cloud environments.

Table 2 summarizes the experiments we performed using one or two instances of the SPECJBB backend. The controller runs on a separate host and is not part of our evaluations (see Figure 12). We used `taskset` to control the CPU affinity of the multi-threaded backend, which we pin to two physical threads in the execution. As a comparison we also run a non-pinned version of the backend on the host (using all available threads) to see the difference in resource utilization. Two dedicated threads are assigned to each VM.

The workload characteristics can be seen in Figure 15, which plots the power estimation of one backend running on one host. One can clearly observe that the estimation of BITWATTS follows the same trend as PowerSpy.

Single node setup. In the literature, applications are usually evaluated in isolated runs. Due to resource sharing, however, process-level estimation becomes more difficult. We further investigate the impact of virtualization as well as interference

Name	Description
Host	
`1BE.2t`	1 backend pinned to 2 threads
`2BE.2t`	2 backends, each pinned to 2 threads
`1BE.4t`	1 backend with 4 threads
VM	
`1BE.1VM.2t`	1 backend, 2 threads, 1 VM
`1BE.2VM.2t`	1 backend, 2 threads, 2 VMs
`2BE.1VM.2t`	2 backends, each 2 threads, 1 VM
Distributed	
`1BE.4t`	2 hosts, 1 backend, 4 threads
`1BE.1VM.2t`	2 hosts, 1 backend, 2 threads, 1 VM

Table 2. Experiments performed using SPECJBB (`BE`: backend, `VM`: virtual machine, `t`: threads).

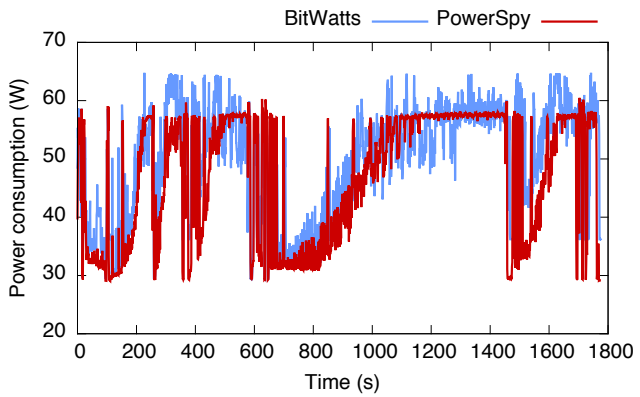

Figure 15. Power consumption during the execution of SPECjbb on the $i3$ with 2 threads.

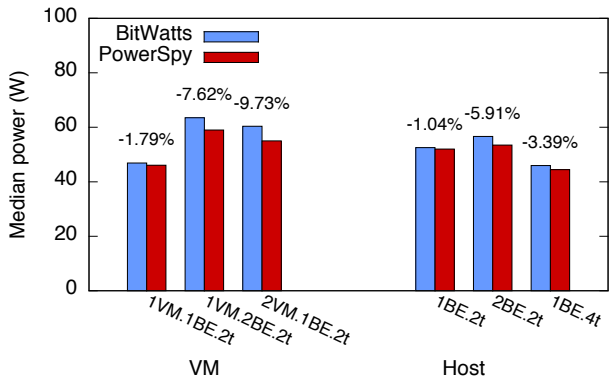

Figure 16. Median power consumption for SPECjbb on $i3$ with different resources assigned to a single or multiple VMs on one host.

of concurrently running applications, first on the host and then in virtual machines. In Figure 16, we report on the median power consumption of the overall SPECJBB run and the median relative error compared to PowerSpy.

On the host, we run once a backend with all available threads and once pinned to 2 threads to ensure that only some of the CPU cores are used. We can see that the accuracy is not influenced if only a part of the CPU is dedicated to a process. In this experiment, we further show that we can monitor two processes at the same time, when running on the host as well as within the VM. Note that we are monitoring both processes separately and only sum up the process power consumption to compare to PowerSpy. As performance counters interfere when more than one process is running, the isolation of the power consumption for each of the process is harder. This is also reflected in the increasing median error if we monitor more than one process at the same time, *e.g.*, when we run 2 backends on the host or within one or two VMs.

In the case of the host running only a single backend, we are underestimating the high-load phases (as can be seen in Figure 15). In general, however, the estimation

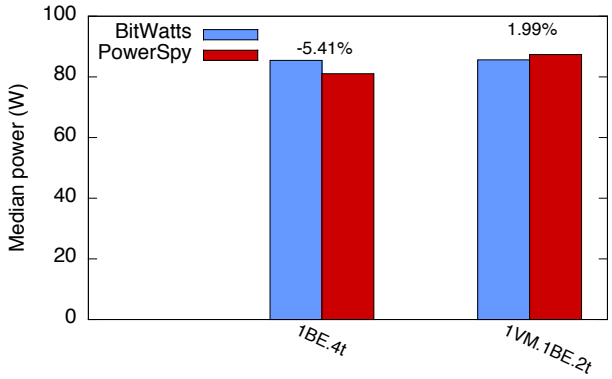

Figure 17. Median power consumption for SPECjbb on $i3$ for a distributed setup, virtualized and non-virtualized.

error is below 10 %. BITWATTS can therefore also estimate real-world applications with load variations and sub-system scenarios when only parts of the CPU are used. We can further observe that virtualization does not cause power consumption overhead, as can be seen in the single VM run with two backends and the two VMs run with one backend each. KVM is hence very power efficient. We can finally see that the backend can use the available resources more efficiently when it has all threads available (see `1BE.4t` vs. `1BE.2t`) as the highest possible throughput in the workload is reached faster than when the backend has limited resources.

Distributed setup. Placing application components in different VMs allow us to execute across multiple hosts. We therefore extend our experiments to a distributed setup, showing that BITWATTS can be applied in realistic data center settings. Experiments were executed on 3 identical servers of type $i3$ as shown in Figure 13.

We first run 1 backend on each host, once with 4 available threads, using BITWATTS. We also execute 1 backend on 2 hosts, each with a VM and 2 threads. The reporting interval to the broker is 1 s. Based on our observations, the contribution of the network interface to the power consumption is very low and is mainly bound to the CPU activity for sending data. Furthermore, the impact of disk access is not covered by the SPECJBB benchmark. At the broker, the values are aggregated and forwarded to the logger, which sums the results and writes them to a file.

The results are shown in Figure 17. As expected, the absolute power consumption increases with running on two hosts because we have to account for both idle values. The median error, however, does not increase as there are instances of BITWATTS on each of the servers and they report the local values to a broker. Furthermore, the single VM experiment shows high accuracy, although underestimating the power consumption. Overall, results are comparable to a single host experiment.

5.4 Summary

We showed that BITWATTS performs well in various situations, notably when scaling VMs and in distributed environments. Power consumption tends to be application dependent. This is the reason why developers start to consider the potential energy footprint of their software. Since the trend is to run software not only locally, but also in data centers and clouds, additional levels of abstraction have to be considered. Based on an application-agnostic power model that supports the power-aware features of modern processors, we deliver a software-defined power meter to estimate the power consumption of distributed and virtualized setups, which are commonly used in cloud environments.

We also demonstrated that our solution is accurate in most cases, even when compared with native information provided by RAPL (see Figure 7).

6. Conclusion

In this paper we presented a middleware toolkit, BITWATTS, for building software-defined power meters. Such software meters provide an accurate alternative to dedicated hardware systems or embedded power counters by estimating power consumption in the small, *i.e.*, at the level of software processes. With BITWATTS we cross the boundaries of virtual environments and provide an estimation of the power consumption of applications running within virtual machines (VMs).

To minimize the estimation error in VMs, BITWATTS needs to deliver accurate power estimation for application-agnostic workloads. We therefore developed a CPU power model that considers the complexity of modern processors, including *multi-cores*, *hyper-threading*, *dynamic voltage/frequency scaling*, and *dynamic overclocking* features that impact power consumption. This power model runs in BITWATTS without hardware support or system alterations to deliver power estimation with a median error of 2 %. To the best of our knowledge, BITWATTS is the first approach to provide such an accurate application-agnostic power model.

Based on this multi-core CPU power model, we proposed a virtual CPU power model that exploits the simplified architecture of virtual processors exposed by the hypervisor to estimate the power consumption of any process running within the VM. The power consumption is forwarded from the host to the VM using an efficient communication channel that connects two instances of BITWATTS. It is noteworthy that the proposed architecture can be scaled to multiple levels of virtualization, depending on the complexity of the environment.

BITWATTS also supports distributed monitoring setups using publish/subscribe middleware to collect and aggregate power measures reported for several application components, in order to deliver a consolidated view of the consumption of a distributed system.

We evaluated the performance of BITWATTS on two processor architectures and in different settings, and we showed that it performs well at different levels of the software stack up to applications.

Power consumption is application dependent, hence developers start to take energy into consideration when programming. As a matter of fact, a growing set of tools are created to provide information about the energy efficiency of software [21, 26]. These tools still require direct access to hardware. Since the trend is to run software not only locally, but also in data centers and clouds, we expect BITWATTS to represent a valuable contribution for researchers, developers, and engineers. The code is freely available as open source.[15]

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under the ParaDIME Project (`www.paradime-project.eu`), grant agreement no. 318693.

References

[1] SPECjbb2013 Design Document. *Standard Performance Evaluation Corporation (SPEC)* (2013).

[2] BEN-YEHUDA, M., DAY, M. D., DUBITZKY, Z., FACTOR, M., HAR'EL, N., GORDON, A., LIGUORI, A., WASSERMAN, O., AND YASSOUR, B.-A. The Turtles Project: Design and Implementation of Nested Virtualization. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Oct. 2010), pp. 423–436.

[3] BERTRAN, R., BECERRA, Y., CARRERA, D., BELTRAN, V., GONZÍLEZ, M., MARTORELL, X., NAVARRO, N., TORRES, J., AND AYGUADÉ, E. Energy Accounting for Shared Virtualized Environments Under DVFS Using PMC-based Power Models. *Future Generation Computer Systems 28*, 2 (2012), pp. 457–468.

[4] BIENIA, C. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, Jan. 2011.

[5] BIRCHER, W. L., AND JOHN, L. K. Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events. In *Proc. of IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)* (Apr. 2007), pp. 158–168.

[6] BOHRA, A., AND CHAUDHARY, V. VMeter: Power Modelling for Virtualized Clouds. In *Proc. of IEEE International Symposium on Parallel Distributed Processing (IPDPSW)* (Apr. 2010), pp. 1–8.

[7] CONTRERAS, G., AND MARTONOSI, M. Power prediction for intel XScale® processors using performance monitoring unit events. In *Proc. of IEEE International Symposium on Low Power Electronics and Design (ISLPED)* (Aug. 2005), pp. 221–226.

[8] COOK, G. How Clean is Your Cloud? Greenpeace, Apr. 2012.

[15] `http://bitwatts.powerapi.org`

[9] HÄHNEL, M., DÖBEL, B., VÖLP, M., AND HÄRTIG, H. Measuring Energy Consumption for Short Code Paths Using RAPL. *ACM SIGMETRICS Performance Evaluation Review 40*, 3 (Dec. 2012), pp. 13–17.

[10] JANACEK, S., SCHRODER, K., SCHOMAKER, G., NEBEL, W., RUSCHEN, M., AND PISTOOR, G. Modeling and approaching a cost transparent, specific data center power consumption. In *Proc. of International Conference on Energy Aware Computing (ICEAC)* (Dec. 2012), pp. 1–6.

[11] KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. A. Virtual Machine Power Metering and Provisioning. In *Proc. of ACM Symposium on Cloud Computing (SoCC)* (June 2010), pp. 39–50.

[12] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. `kvm`: the Linux Virtual Machine Monitor. In *Proc. of the Linux Symposium* (June 2007), vol. 1, pp. 225–230.

[13] KNAUTH, T., AND FETZER, C. DreamServer: Truly On-Demand Cloud Services. In *Proc. of ACM SIGOPS International Conference on Systems & Storage (SYSTOR)* (June 2014), pp. 1–11.

[14] KOLLER, R., VERMA, A., AND NEOGI, A. WattApp: An Application Aware Power Meter for Shared Data Centers. In *Proc. of ACM International Conference on Autonomic Computing (ICAC)* (June 2010), pp. 31–40.

[15] KREVAT, E., TUCEK, J., AND GANGER, G. R. Disks Are Like Snowflakes: No Two Are Alike. In *Proc. of USENIX conference on Hot topics in Operating Systems (HotOS)* (May 2011), pp. 14–14.

[16] KRISHNAN, B., AMUR, H., GAVRILOVSKA, A., AND SCHWAN, K. VM Power Metering: Feasibility and Challenges. *ACM SIGMETRICS Performance Evaluation Review 38*, 3 (Jan. 2011), pp. 56–60.

[17] LI, T., AND JOHN, L. K. Run-time Modeling and Estimation of Operating System Power Consumption. *ACM SIGMETRICS Performance Evaluation Review 31*, 1 (June 2003), pp. 160–171.

[18] LIM, M. Y., PORTERFIELD, A., AND FOWLER, R. J. SoftPower: Fine-Grain Power Estimations Using Performance Counters. In *Proc. of ACM International Symposium on High Performance Distributed Computing (HPDC)* (June 2010), pp. 308–311.

[19] MCCULLOUGH, J. C., AGARWAL, Y., CHANDRASHEKAR, J., KUPPUSWAMY, S., SNOEREN, A. C., AND GUPTA, R. K. Evaluating the Effectiveness of Model-Based Power Characterization. In *Proc. of USENIX Annual Technical Conference (ATC)* (June 2011), pp. 12–12.

[20] NOUREDDINE, A., BOURDON, A., ROUVOY, R., AND SEINTURIER, L. Runtime Monitoring of Software Energy Hotspots. In *Proc. of the IEEE/ACM International Conference on Automated Software Engineering (ASE)* (Sept. 2012), pp. 160–169.

[21] NOUREDDINE, A., ROUVOY, R., AND SEINTURIER, L. Unit Testing of Energy Consumption of Software Libraries. In *Proc. of ACM Symposium On Applied Computing* (Mar. 2014), pp. 1200–1205.

[22] ORGERIE, A.-C., ASSUNCAO, M. D. D., AND LEFEVRE, L. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Computing Surveys (CSUR) 46*, 4 (Apr. 2014), pp. 47:1–47:31.

[23] POWELL, M. D., BISWAS, A., EMER, J. S., MUKHERJEE, S. S., SHEIKH, B. R., AND YARDI, S. CAMP: A Technique to Estimate Per-Structure Power at Run-time using a Few Simple Parameters. In *Proc. of IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Feb. 2009), pp. 289–300.

[24] SANDERSON, D. *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure*. O'Reilly Media, Inc., Dec. 2009.

[25] SHEN, K., SHRIRAMAN, A., DWARKADAS, S., ZHANG, X., AND CHEN, Z. Power Containers: An OS Facility for Fine-grained Power and Energy Management on Multicore Servers. In *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (Apr. 2013), pp. 65–76.

[26] STERLING, C. Energy Consumption tool in Visual Studio 2013, July 2013.

[27] STOESS, J., LANG, C., AND BELLOSA, F. Energy Management for Hypervisor-Based Virtual Machines. In *Proc. of USENIX Annual Technical Conference (ATC)* (June 2007), pp. 1–14.

[28] THE CLIMATE GROUP. SMART 2020: Enabling the low carbon economy in the information age, 2008.

[29] VERMA, A., AHUJA, P., AND NEOGI, A. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Proc. of ACM/IFIP/USENIX Middleware Conference* (Dec. 2008), Springer, pp. 243–264.

[30] VERSICK, D., WASSMANN, I., AND TAVANGARIAN, D. Power Consumption Estimation of CPU and Peripheral Components in Virtual Machines. *ACM SIGAPP Applied Computing Review 13*, 3 (Sept. 2013), pp. 17–25.

[31] WANG, S., CHEN, H., AND SHI, W. SPAN: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems 1*, 1 (2011), pp. 23–34.

[32] ZHAI, Y., ZHANG, X., ERANIAN, S., TANG, L., AND MARS, J. HaPPy: Hyperthread-aware Power Profiling Dynamically. In *Proc. of USENIX Annual Technical Conference (ATC)* (June 2014), pp. 211–217.