Power Consumption Models for Multi-Tenant Server Infrastructures

MATTEO FERRONI, ANDREA CORNA, ANDREA DAMIANI, and ROLANDO BRONDOLIN, Politecnico di Milano
JUAN A. COLMENARES, Samsung Research America
STEVEN HOFMEYR, Lawrence Berkeley National Laboratory
JOHN D. KUBIATOWICZ, University of California Berkeley
MARCO D. SANTAMBROGIO, Politecnico di Milano

Multi-tenant virtualized infrastructures allow cloud providers to minimize costs through workload consolidation. One of the largest costs is power consumption, which is challenging to understand in heterogeneous environments. We propose a power modeling methodology that tackles this complexity using a divide-and-conquer approach. Our results outperform previous research work, achieving a relative error of 2% on average and under 4% in almost all cases. Models are portable across similar architectures, enabling predictions of power consumption before migrating a tenant to a different hardware platform. Moreover, we show the models allow us evaluate colocations of tenants to reduce overall consumption.

CCS Concepts: •**Computing methodologies** →**Modeling methodologies;** •**Hardware** →**Power estimation and optimization;** •**Software and its engineering** →*Virtual machines;*

ACM Reference format:
Matteo Ferroni, Andrea Corna, Andrea Damiani, Rolando Brondolin, Juan A. Colmenares, Steven Hofmeyr, John D. Kubiatowicz, and Marco D. Santambrogio. 2010. Power Consumption Models for Multi-Tenant Server Infrastructures. *ACM Transactions on Architecture and Code Optimization* 9, 4, Article 39 (March 2010), 21 pages.
DOI: 0000001.0000001

1 INTRODUCTION

Virtualization has become an extremely important tool for organizing computer systems [26, 34]. It provides a clean separation of software development concerns from the underlying hardware platform, allowing multiple tenant applications to share physical resources while fulfilling needs for isolation and security [9, 30, 39].

The abstraction provided by virtualization fosters *heterogeneity* from both the virtual tenants and the underlying infrastructure. On the one hand, virtual tenants may be intrinsically different from one another due to different workload limitations (i.e., they can be memory-bound, I/O-bound and/or CPU-bound) and evolving load patterns (e.g., algorithmic phases). On the other hand, each physical host may exhibit different performance and power characteristics from other hosts (even supposedly

Author's addresses: Matteo Ferroni, Rolando Brondolin, Marco D. Santambrogio {matteo.ferroni, rolando.brondolin, marco.santambrogio}@polimi.it; Andrea Damiani, Andrea Corna {andrea.corna, andrea3.damiani}@mail.polimi.it; Juan Colmenares juan.col@samsung.com; Steven Hofmeyr shofmeyr@lbl.gov; John D. Kubiatowicz, kubitron@cs.berkeley.edu.
 XXXX-XXXX/2010/3-ART39 $15.00
DOI: 0000001.0000001

identical ones), given the same set of tenants. It is easy to see how a virtualization infrastructure requires sophisticated approaches to resource allocation and accounting, a requirement that can quickly become intractable as the number of virtual tenants per host increases [38].

Things become more difficult if we want to consider also *power consumption*, as data centers providers aim to reduce it as much as possible to decrease operating costs and to improve system reliability. Even though the performance-per-watt ratio has been constantly rising, the total power drawn is hardly decreasing and recent trends suggest that the cost of the energy consumed by a server during its lifetime will probably exceed the hardware cost in the near future [13].

Given the strong correlation with live operating costs, power consumption *consolidation* through tenant colocation and migration becomes critical for the Cloud Computing paradigm, which delivers computing services as a utility in a "pay-as-you-go" manner [10]. Much recent work [18, 19] focuses on the problem of power optimization under performance constraints, defined in terms of Service Level Agreements (SLAs) between the service provider and the tenant. For most of these approaches, an accurate *model* of tenants' behavior is a first step to guaranteeing tenants' requirements and optimizing physical resource utilization.

1.1 Contribution

In this article, we present a power modeling methodology that tackles the complexity that comes from heterogeneity through the identification of *working regimes*: the idea is to identify the working states of the system, and build a model for *each* of them. This is different from the typical approach in the literature, which uses static assumptions about hardware components to attempt to build a *single* comprehensive model of the behavior of the system [13]. The input of the model consists in a set of hardware event traces, which represent the link between hardware utilization and the overall power consumption, i.e., the output of the model. The proposed approach is completely *data-driven*, because no knowledge about the internal components of the system is required.

We demonstrate the accuracy of our methodolgy with experiments in a multi-tenant virtualized infrastructure based on the Xen hypervisor [11]. Our results show a relative error with respect to the measured dynamic power consumption of 2% on average, and under 4% in most cases, which is more accurate than previous results in the literature [13, 15, 41]. We also show that the same model can be exploited to make predictions about the impact of power consumption when migrating a domain (i.e. a Virtual Machine (VM) managed by the Xen hypervisor) to a different hardware platform, with little loss in accuracy.

Finally, we show how these power models can be used to produce a metric of power efficiency that makes it possible to compare different colocations of tenants, as long as there exists a runtime monitoring system that is able to attribute hardware events to each domain. A cluster-level scheduler could use that metric to evaluate which particular colocation leads to better consolidation from a power consumption perspective.

All the code developed to monitor the system, build the power models, run the tests and validate the approach is available as open source at https://bitbucket.org/necst/2017-powermodels, and we expect it become a reference for future work in the field of virtualized, power-aware systems.

1.2 Roadmap

The rest of this article is organized as follows. Section 2 explores how different workload types influence system power consumption, baring the limitations of most common power modeling approaches. Section 3 discusses the related work in the field. Section 4 presents the proposed methodology, from the high-level logic flow to relevant implementation details. Section 5 describes

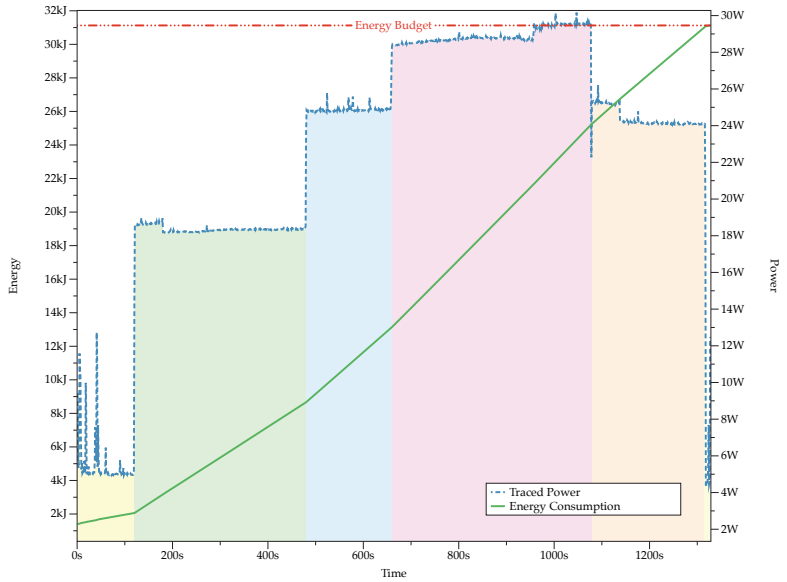


Fig. 1. Example of a power trace obtained by running a benchmark program that goes through the following phases: (1) idle, (2) data load (moderately I/O-intensive), (3) data preparation (memory-intensive), (4) computation (CPU-intensive), and (5) data store (highly I/O-intensive).

the experiments conducted to validate our methodology and discusses the results. Section 6 concludes the paper and discusses our future work.

2 MOTIVATING EXAMPLE

For consolidation techniques to be more effective, they need to estimate the impact of workloads on power consumption. Workloads strongly influence the power consumed by the systems on which they run [15, 40, 41]. We thus start our study by exploring how workload heterogeneity affects power consumption.

We built a synthetic benchmark that reproduces a sequence of workloads with different characteristics. It comprises various execution phases; they are: (1) *idle phase*: the benchmark has not started yet; (2) *data load phase*: a large amount of data is loaded into main memory (I/O-intensive task); (3) *data preparation phase*: the data in memory is prepared to be processed (memory-intensive task); (4) *computation phase*: the data is processed (CPU-intensive task); (5) *data store phase*: the results are written to persistent storage (I/O-intensive task); and (6) *idle phase*: the benchmark terminates. Figure 1 shows a power trace of a server (equipped with a 2.8-GHz quad-core Intel Xeon E5-1410 processor and 32GB of RAM) while running the benchmark. Power traces were obtained from a power meter connected to the server. Distinct phases are reported with different colors to highlight their differences in power consumption.

We ran the benchmark multiple times and collected traces of hardware events via architectural Performance Monitoring Counters (PMCs). These traces represent low-level metrics of hardware utilization and have been extensively used in the literature to build power models [13]. Since the early work in [15], some researchers have used hardware events as input to linear models to explain the behavior of generic workloads (e.g., [22]). Others later realized that such simple models are not enough to tackle more complex scenarios [20]. Most recent work introduces the concept of

Table 1. Performance of the ARX models in terms of root mean square error (RMSE) and mean relative error with respect to measured dynamic power consumption. The ARX models are built starting from the input features selected for each model class, as detailed in Section 2.

Working State	Model Class A		Model Class B		Model Class C	
	RMSE	Relative Error	RMSE	Relative Error	RMSE	Relative Error
(1) Idle	± 17.63 W	35.56%	± 16.44 W	32%	± 17.68 W	35%
(2) Moderately I/O-intensive	± 4.7 W	9.4%	± 5.86 W	11.7%	± 7.17 W	14%
(3) Memory-intensive	± 19.11 W	38%	± 34.54 W	70%	± 18.7 W	37%
(4) CPU-intensive	± 0.44 W	0.08%	± 0.6W W	1.2%	± 0.42 W	0.08%
(5) Highly I/O-intensive	± 2.98 W	5.9%	± 38.57 W	77%	± 3.29 W	6.5%
Average	± 8.97 W	17.79%	± 19.20 W	38.38%	± 9.45 W	18.52%

"intensiveness" and uses separate linear or AutoRegressive with eXogenous input (ARX) models for different classes of workloads [40, 41].

Inspired by this most recent work, we built different ARX models for each "prevailing intensity" we observed in our synthetic benchmark. As shown in Figure 1, the system goes through five different working states: (1) idle, (2) moderately I/O-intensive, (3) memory-intensive, (4) CPU-intensive, and (5) highly I/O-intensive. We resorted to the literature to choose the subset of hardware events that best correlate with power consumption [15, 41]; we chose four events: `INST_RET` (number of instructions at retirement), `UNHALTED_CLOCK_CYCLES` (number of cycles where a core is not halted), `LLC_REF` (number of requests to the last level cache), and `LLC_MISS` (number of requests to the last level cache resulting in misses). In order to find the most satisfactory subset of input features for these preliminary models, we tried all the possible combinations ($2^4 - 1 = 15$) of these events. We selected the best performing combinations and used them to build the following classes of models:

- *Model Class A*: *input* $\in$ {`INST_RET`, `UNHALTED_CLOCK_CYCLES`, `LLC_REF`, `LLC_MISS`}
- *Model Class B*: *input* $\in$ {`INST_RET`, `UNHALTED_CLOCK_CYCLES`, `LLC_REF`}
- *Model Class C*: *input* $\in$ {`UNHALTED_CLOCK_CYCLES`, `LLC_REF`}

Note that each model class receives a different set of events as input. We built a separate ARX model for each pair <*Model Class, Working State*>, and Table 1 reports the models' performance in terms of Root Mean Square Error (RMSE) and mean relative error.

Our exploratory results yield the following observations. First, as illustrated in Figure 1, distinct workload types make the hosting system operate in different working states and consume different amounts of power. Second, Table 1 shows that there is a good correlation between some hardware events and system power consumption, as often reported in literature. It is then reasonable to use selected hardware events to build power models. But choosing the right hardware events is key to modeling accuracy. While the models built for the CPU-intensive working state predicted the system's power consumption reasonably well, the models built for the memory-intensive working state and the idle state performed rather poorly (with relative errors above 30% and up to 70%) and so did the Class B model for the highly I/O-intensive working state (with relative error of 77%). It is then clear the need for a methodology to automatically selecting the set of hardware events that produce high modeling accuracy. Finally, it is also evident that a single ARX model, in general, cannot accurately predict the power consumption of a system that runs different types of workloads. In Section 4, we present a power modeling methodology that takes these observations into account and produces accurate results.

3 RELATED WORK

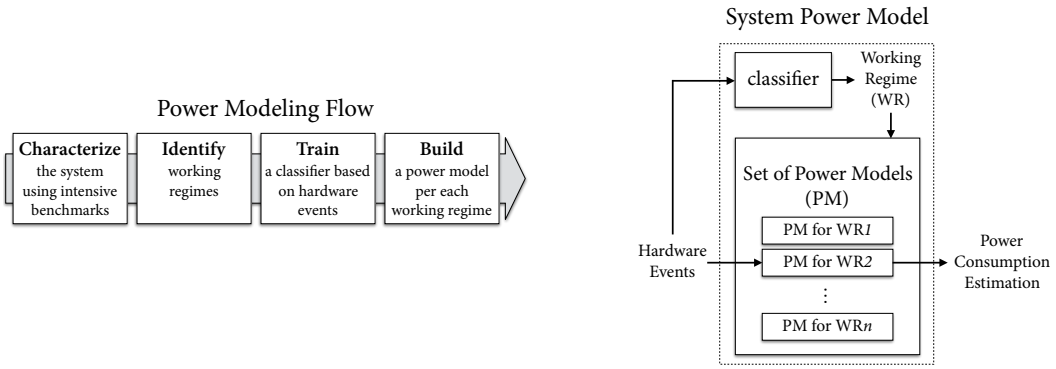
There exists an extensive body of research on power models for physical machines. Studies in this field started around 2007 with the work by *Bircher and John* [15]. They aimed at modeling the power consumption of a physical machine and not only of its microprocessor, which was the focus of earlier work [12, 16, 27, 31, 32]. Using a subset of the embryonic PMCs of the Intel Pentium IV processor, they managed to obtain linear regression models of the energy consumed by each subsystem, with less than 10% of relative error. Most importantly, this early work foresaw the necessity for power accounting over virtual tenants.

Few years later, the Cloud Computing research community started producing interesting methods to power modeling in virtualized environments. *Kansal et al.* proposed *Joulemeter* [28]. The main reason for developing this power metering tool was that, in traditional systems, the visibility of the Operating System (OS) over the workloads had always been exploited to *"make automated and manual power management decisions,"* but with the *isolation property* enforced by virtual machines, such visibility got disrupted. The authors pioneered an energy inference methodology based on resource usage; they built models for each hardware subsystem using only software performance counters in the hypervisor, with an absolute error of 5W. However, the main caveat of this work is the non-automatic generation of power models, undermining the applicability of the study to different machines and architectures.

Bertran et al. [14] leveraged Performance Monitoring Counter (PMC) measurements to build linear power models. This work achieves less than 5% prediction error and shows that Dynamic Voltage Frequency Scaling (DVFS) does not affect model precision. The proposed models are based on micro-architectural features of the processor that are characterized depending on selected PMCs and thus tied to the system under study. Moreover, the paper focuses on models that are benchmark-independent. By contrast, we seek to enhance model precision and enable model portability (still using PMCs) by building not only generic models, but also specific models for each working regime of each tenant.

With a goal similar to ours, *Gu et al.* [25] developed a tree regression based approach for VM power modeling. They partition the dataset used for modeling with respect to the input features (i.e., CPU, memory and I/O usage). They then organize the model obtained from each partitioned dataset using a tree that is traversed to calculate the coefficients to be used for a given input measurement. This tree-based approach is able to identify different working conditions for distinct workloads, but the experimental results only show up to 90% of accuracy when considering dynamic power. In comparison, our methodology yields better accuracy in all our experiments.

A more complete study on VM power modeling is given by *Yang et al.* [41], in which they introduce *iMeter*, an integrated VM power model based on performance profiling. This work is one of the first to present a highly systematic approach, which comprises the following steps: (1) benchmark selection, where the authors use the *NASA Parallel Benchmark* suite [21, 40] plus *IOzone* [7] and *Cachebench* [35] to stress the VMs with various workloads; (2) PMC trickle-down selection, necessary for reducing the solution space; (3) establishment of a measurement baseline, obtained by running VMs on a physical machine; (4) systematic selection of performance counters, exploiting principal component analysis (PCA) and varimax rotation for further model order reduction; (5) modeling, leveraging *support vector regression*; and (6) evaluation, where hierarchical clustering is applied to assimilate similar conditions. iMeter was able to reach a relative error of about 5%. iMeter builds just one model for each VM, while our methodology tries to leverage a better understanding of the working conditions of each tenant. In the present work, we show how to improve the modeling accuracy through working regime identification, thus raising the bar in the field of power modeling methodologies for multi-tenant server infrastructures.



(a) Steps for generating a system power model.

(b) Structure of a system's power model.

Fig. 2. Overview of the proposed power modeling methodology based on work regime identification.

4 PROPOSED METHODOLOGY

This section presents a power modeling methodology that tackles the complexity associated with workload heterogeneity, discussed in Section 2, through the identification of working regimes. A *working regime* is an operating state in which the modeled system, stressed by a single workload or a mix of simultaneous workloads, consumes a certain amount of power. Each operating state is characterized by value ranges from selected hardware events. Thus, working regimes are conceptually decoupled from (and only loosely related to) workloads and application execution phases – a single workload or a mix of workloads may both make the system operate under the same working regime and thereby draw similar amount of power.

As shown in Figure 2b, our methodology builds a system's power model that combines individual models, each built for a distinct working regime observed in the system. The methodology comprises the following steps (see Figure 2a):

(1) *System characterization*: The hardware is stressed with a representative set of benchmarks. The benchmarks have to cover numerous diverse computational patterns in order to observe the different working regimes that characterize the system.
(2) *Identification of working regimes*: We adopt a data-driven approach to regime identification because, as shown in Section 2, the "prevailing intensities" [40, 41] of distinct workloads may lead to dramatic disparities between power models and actual system power consumption. We propose the use of a clustering technique to identify working regimes from observed power traces.
(3) *Classifier training*: Once the regimes are identified, they, along with hardware metrics, become the *training set* for the working regime classifier. It will then be used to identify the system's work regime at runtime.
(4) *Model building per working regime*: In this final step an ARX power model is built for each working regime in the training set. These models, in conjunction with the classifier, constitute the entire power model of the system, as illustrated in Figure 2b.

At runtime, hardware events are collected at a certain rate. The classifier determines the current working regime of the system from the trace of hardware events and chooses the ARX power model accordingly. Then, the hardware events are given to the selected model to produce an estimate of the system's power consumption ($\hat{P}_{single}$).

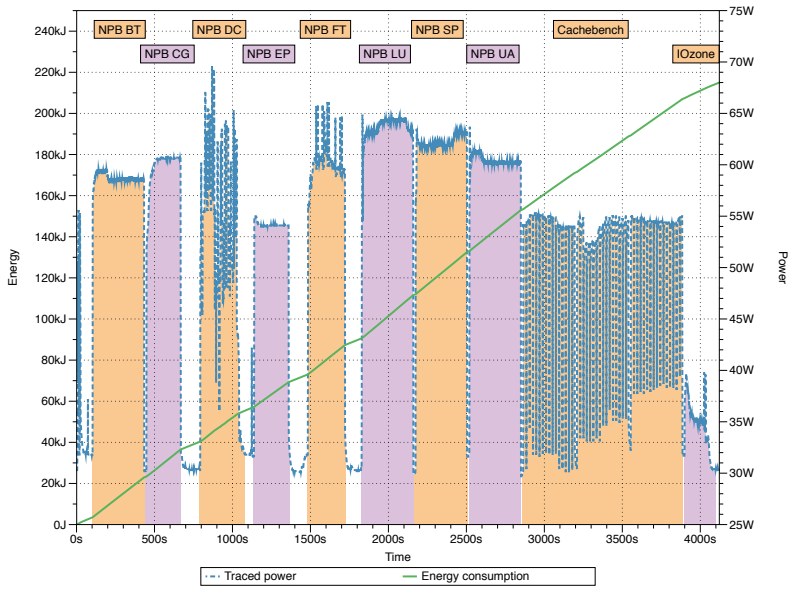


Fig. 3. Power and energy trace obtained while running the chosen benchmarks on our server machine SRV1.

Next, we discuss the adopted techniques in each step of our methodology and provide details of their implementation.

4.1 System Characterization

In this work, we use a set of micro-benchmarks to stimulate the hardware, narrowing down to single aspects (e.g., CPU performance and memory latency). The goal is to make a physical machine exhibit as many as possible (ideally all) of its different regimes of power consumption.

We chose the NAS Parallel Benchmarks (NPB) suite [21], given its ability to stress the CPU and RAM with a mix of eight highly targeted benchmarks [41]. In addition, we adopted Cachebench [35] and IOzone [7] to stress the cache hierarchies and file I/O by transferring large streams of data. Figure 3 shows the power energy trace obtained from a sample run of the selected benchmarks on our test server machine SRV1, with a 2.8-GHz quad-core Intel Xeon E5-1410 processor, Simultaneous MultiThreading (SMT) enabled, and 32GB of RAM.

Note that none of the selected benchmarks stresses network I/O. Characterizing virtualized network I/O in terms of power consumption requires, besides various representative benchmarks, a special configuration. Servers purposely deployed to host VMs often offer improved virtualized network performance by exploiting features such as multi-queue NICs, VMs with direct access to NIC queues, and specialized privileged drivers. Since identifying power regimes related to network activity is not essential to evaluate the proposed methodology, we leave it as a future work.

4.2 Identification of Working Regimes

The goal here is to identify a reasonable number of working regimes from the power traces obtained in the previous step (Section 4.1). To this end, we adopt the following procedure: (1) we estimate a *probability density function* from the data set of power measurements using the Kernel Density Estimation (KDE) method [36], (2) we find all the local minima of the estimated probability density

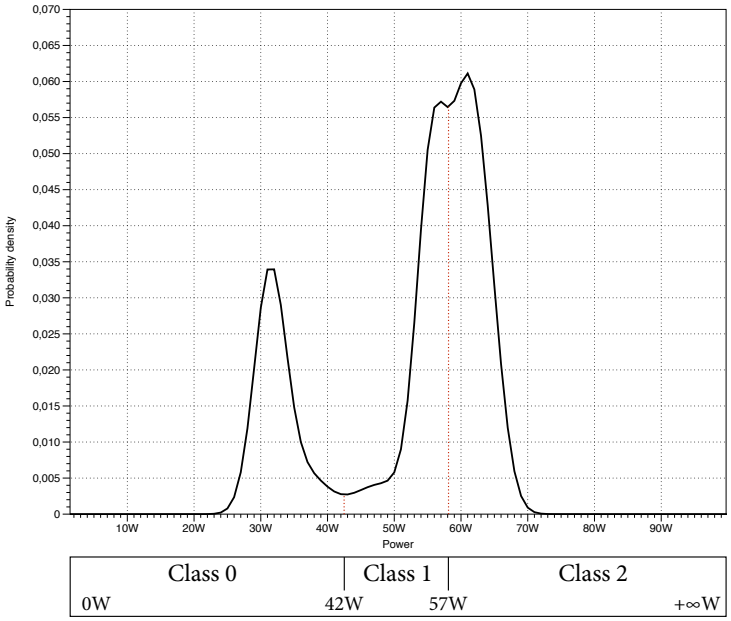


Fig. 4. Example of the identification of working regimes on our server machine SRV1. The probability density function's local minima demarcate the classes of power consumption.

function, and (3) we split the data set into *intervals* using the local minima as boundaries. Note that the procedure is applied to a single dimension in the data set – namely, the power consumption values. The rationale behind it is that each identified interval will contain highly "packed" values indicating the presence of a steady operating state (or working regime) that lies inside the interval.

KDE [36] is a non-parametric method to estimate the probability density function of a sampled random variable. Let $(x_1, \ldots, x_n)$ be an independent and identically distributed sample drawn from a distribution of unknown probability density function f. According to KDE, the estimator of f is:

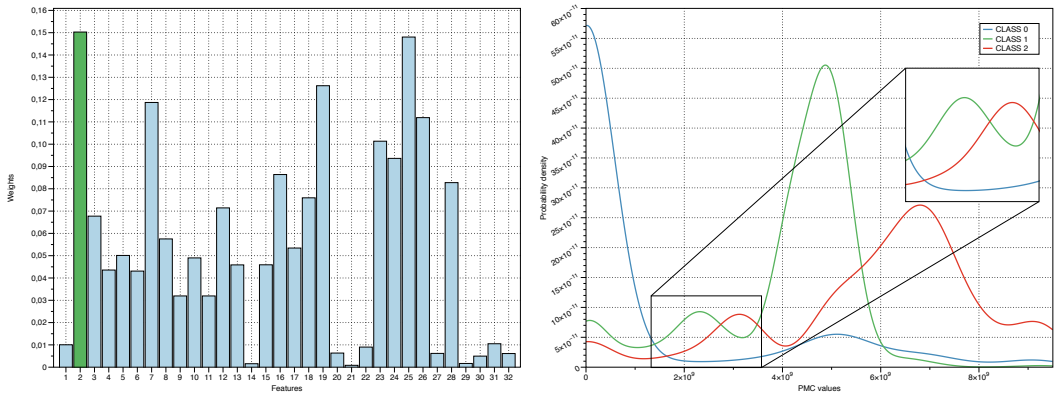
$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{1}$$

for a given *kernel function* $K(\bullet)$, a non-negative weighting function that integrates to one and has zero mean, and $h > 0$, a smoothing parameter known as *bandwidth*. We adopt the Gaussian kernel function:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{2}$$

as it best balances computational efficiency and accuracy.

Figure 4 exemplifies an automatic run of the above procedure to identifying working regimes on SRV1. First, KDE produces the probability density function from a power trace. Next, the two local minima of the probability density function are obtained – one at 42W and the other at 57W. Acting as boundaries, they defined three intervals indicated at the bottom of the figure. Finally, the power data is split according these boundaries. We notice that the local minimum at 57W in Figure 4 is very close to the peaks surrounding it, so it is reasonable to merge the intervals denoted as "Class 1" and "Class 2" into a single interval. It is possible to extend the procedure in order to detect this



(a) Estimated weights for the input features. Features with higher weights are more important for classification.

(b) Probability distributions of Feature 2 (INST_RET) for each regime class, with the range of uncertainty highlighted.

Fig. 5. Results of the first iteration of the ReliefF algorithm in our running example on server machine SRV1. Feature 2 (INST_RET hardware event) is a good candidate feature, but not enough by itself to perform a good classification.

type of situations and allow the user to refine the intervals or even do it automatically. However, in this we found no need to do so.

4.3 Working Regime Classifier

Once we obtain the intervals of power consumption that identifies each working regime, we need to infer this classification directly from the *input features*, i.e., the trace of hardware events collected from the selected benchmarks in Section 4.1. For this purpose, we use *ReliefF* [29], a distance-based supervised classification algorithm with good trade-off between accuracy and efficiency [17].

Figure 5a shows the weights of the input features computed by a first round of ReliefF in our running example. Feature names (i.e., hardware event codes) are omitted for readability. Feature 2, corresponding to the INST_RET event, gets the highest weight and it is thus the most important feature for classification. However, this feature alone is not always sufficient to discriminate among the three regime classes identified. Figure 5b shows the probability distribution of this feature for the regime classes, reconstructed using KDE. The figure highlights that there exists at least a range of values for which it is not possible to perform a good classification.

This uncertainty can be reduced by another iteration of ReliefF with the remaining features targeting the problematic range. We can see in Figure 5a that Feature 25 (MEM_LOAD_UOPS_RET_L1_HIT event) can help us obtain a good discrimination among the uncertain classes in ranges where Feature 2 is not enough to perform an accurate classification. This process can be repeated until enough features have been chosen to ensure good classification performance. For each selected feature, we have as outcome a range of PMC values that can be used to identify a different working regime of the system at runtime. Table 2 shows the result for the running example of this section.

4.4 Power Models for Working Regimes

After obtaining the PMC value ranges that define the classifier for the target system's working regimes, we can build a power model for each regime. We adopt *AutoRegressive models with eXogenous inputs* (ARX models) due to their generality and simplicity.

Table 2. Set of rules for the working regime classifier in our running example.

	INST_RET	MEM_LOAD_ UOPS_RET_L1_HIT
CLASS 0	$[0, 1.235 \times 10^9]$	
CLASS 1	$[3.61 \times 10^9, 5.58 \times 10^9)$	
	$(1.235 \times 10^9, 3.61 \times 10^9)$	$[2.36362 \times 10^8, 5.672 \times 10^8]$
CLASS 2	$[5.58 \times 10^9, +\infty)$	
	$(1.235 \times 10^9, 3.61 \times 10^9)$	$[0, 2.36362 \times 10^8) \cup (5.672 \times 10^8, +\infty)$

Table 3. Coefficients for the output feature (rapl_pkg) and the exogenous input features (UNHALTED_CORE_CYCLE, INST_RET, LLC_REF, and L1_HIT) of the ARX models in our running example on server machine SRV1. Each model corresponds to a separate working regime.

Features	Working Regime 0 (CLASS 0)	Working Regime 1 (CLASS 1)	Working Regime 2 (CLASS 2)
rapl_pkg ($\alpha_{r,1}$)	1.000133311	0.999987546	1.000051151
UNHALTED_CORE_CYCLE ($\alpha_{x,1,0}$)	$1.585235081E-9$	$5.907284775E-9$	$4.392083655E-7$
INST_RET ($\alpha_{x,2,0}$)	$6.966616339E-9$	$-1.41164093E-10$	0.000482357
LLC_REF ($\alpha_{x,3,0}$)	$1.586185546E-7$	$3.31791638E-9$	0.001427735
L1_HIT ($\alpha_{x,4,0}$)	$-8.587632240E-9$	$1.08671292E-9$	−0.002176198

We use the data recorded while running the benchmarks in the system characterization step (Section 4.1). The dataset comprises a time series of power measurements and hardware event counts for a succession of time intervals. The classifier allows us to split the dataset into batches, i.e., a group of samples related to the same working regime. The *training set* for each model is then obtained by grouping together data batches for the same regime.

For the ARX model of a working regime, power consumption is the *output feature* f_r and the hardware event counts are the *exogenous input features* $f_{x,i}$. An ARX model includes the autoregressive component, which is recursively defined as the linear combination of past values of the output (f_r), and the exogenous component, which is the linear combination of past values of the exogenous inputs ($f_{x,i}$). A general ARX model is shown below:

$$\begin{aligned} f_r(t) = {} & \alpha_{r,1} f_r(t-1) \; + \; \ldots \; + \; \alpha_{r,ar} f_r(t-ar) \; + \\ & + \; \alpha_{x,1,0} f_{x,1}(t) \; + \; \ldots \; + \; \alpha_{x,1,ex} f_{x,1}(t-ex) \; + \\ & + \; \ldots \; + \\ & + \; \alpha_{x,n,0} f_{x,n}(t) \; + \; \ldots \; + \; \alpha_{x,n,ex} f_{x,n}(t-ex) \end{aligned} \tag{3}$$

ARX models are built with specific past autoregressive and exogenous values. The number of previous samples used to estimate the current value of the output variable is referred to as *delay*. Throughout the course of this work, we explored different delay values for the autoregressive and exogenous components. We found that a one-step delay for the autoregressive component ($ar = 1$) and zero delay for the exogenous inputs ($ex = 0$) generally lead to good modeling precision. Hence, we adopt the following simplified ARX formulation derived from Equation (3):

$$f_r(t) = \alpha_{r,1} f_r(t-1) \; + \; \alpha_{x,1,0} f_{x,1}(t) \; + \; \ldots \; + \; \alpha_{x,n,0} f_{x,n}(t) \tag{4}$$

We are then implicitly assuming that the power consumption of the system is highly dominated and depends almost entirely on its current load.

Table 3 presents the coefficients of the three ARX models, one per identified working regime, generated from this section's running example. The output feature `rapl_pkg` is the socket power consumption reported by Intel RAPL interface [37], while the exogenous input features are the following hardware event counters: (1) `UNHALTED_CORE_CYCLE` (number of core clock cycles whenever the logical processor is in C0 state – i.e., not halted), (2) `INST_RET` (number of instructions at retirement), (3) `LLC_REF` (number of requests to the last level cache), and (4) `L1_HIT` (number of load hits in nearest-level data cache). Note that the features chosen for the models may be different from those used by the working regime classifier. We also observe from Table 3 that the autoregressive component is the dominant component across the three models, and the exogenous components have a stronger influence under the working regime 2 than under the regimes 1 and 3.

4.5 Implementation Details

The proposed methodology requires: (1) a tool that collects hardware events and accounts them to tenants, and (2) a modeling pipeline that automatically parses, cleanses and processes raw traces to produce power models.

We meet the first requirement by using hypervisor-level instrumentation to monitor context switches between virtual tenants [23]. At a given context switch, we configure the PMC registers to store specific hardware event counts for the tenant that is about to run. This is done on every CPU (i.e., physical core or hardware thread) of the system, as each tenant may have multiple virtual CPUs (VCPUs). Socket-level energy measurements are read via Intel Running Average Power Limit (RAPL) interface [37] at each context switch, too. At the next context switch, the PMC are read and their values attributed to the tenant that was running. The counters are then cleared for the next tenant to run. Finally, PMC values are aggregated by tenant and stored on disk.

For the second requirement, we used MARC, a Model-as-a-Service toolchain for modeling resource consumption [24]. MARC, which stands for *Model Analysis for Resource Consumption*, supported the entire process of exploring, producing and refining the models used in this work. It helped us correct inconsistencies (e.g., counter overflows) in hardware event traces and perform other data conditioning tasks, as well as associate portions of time-series data with specific working regimes. We also used it to build and evaluate the ARX models for the working regimes. Thanks to MARC we had no need to implement auxiliary scripts for data analysis. As stated before, all our code is available as open source at https://bitbucket.org/necst/2017-powermodels.

5 EVALUATION

In this section, we evaluate our power modeling methodology. First, we assess the precision of the power models produced with the methodology (Section 5.2). We compare *generic models*, which are workload-agnostic, against *workload-specific models*. Second, we study how portable generic models are across different machines (Section 5.3). Finally, we show how the models allow us to compare different colocations of virtual tenants based on predicted power consumption (Section 5.4).

5.1 Experimental Setup

We use different system configurations to validate our methodology under a reasonable degree of heterogeneity. Below we describe the machines and workloads used in our experiments.

Hardware Platforms. It is quite common in data centers that machines with different characteristics are part of the same cluster, e.g., as batches of new servers gradually replace old ones. We thus conduct our evaluation on three distinct machines:

- **SRV1**: a Dell PowerEdge T320 equipped with an Intel 2.8GHz Xeon E5-1410 processor (4 cores and 8 hyper-threads) and 16GB of DDR3 RAM, which represents a recent low-range server.
- **SRV2**: a Dell PowerEdge T630 with two Intel 2.3GHz Xeon E5-2650 v3 processors (10 cores and 20 hyper-threads each) and 128GB of DDR4 RAM, which represents a recent mid-range server.
- **WRK**: a Dell OptiPlex 990, equipped with an Intel 3.4GHz Core i7-2600 processor (4 cores and 8 hyper-threads) and 8GB of DDR3 RAM. It is an old, but still popular, general-purpose desktop machine that we include in our evaluation to show the broad applicability of our methodology.

Experiments are run with and without SMT. Intel TurboBoost is disabled in all the machines, as it tends to cause unpredictable performance behavior [8, 33].

In addition, we report power measurements obtained with the *Intel RAPL interface* [37] and *WattsUp power meters* [6]. The RAPL interface, available on Intel Sandy Bridge and later processors, offers a set of Model Specific Registers (MSRs) able to produce CPU power measurements with ~1ms granularity. We specifically use the registers that report whole socket power consumption (i.e., PKG MSRs). A WattsUp power meter, on the other hand, is an external device installed between the power source and the test machine; it logs power consumption approximately every second.

Test Workloads. We use a representative set of benchmarks widely adopted in cloud environments; they are listed below, grouped by their predominant observed behavior:

- *Compute-intensive*: the Support Vector Machine (SVM) and PageRank algorithms implemented on Apache Spark [2], an engine for big data processing, and a stress benchmark for Redis [5], an in-memory data store.
- *Memory-intensive*: a benchmark for MySQL [4] and another for Cassandra [1] to stress a SQL and a NoSQL database system.
- *I/O-intensive*: a benchmark for FFmpeg [3], an audio-video processing tool suite.

Each test workload executes in a separate Xen domain. Each domain runs a minimal Debian Linux distribution and is assigned two virtual CPUs (VCPUs). These VCPUs are pinned onto two cores in order to improve performance stability and limit scheduling overhead.

Note that we only run a subset of the test workloads (i.e., FFMPEG, Redis, and MySQL) on the WRK machine due to its limited capabilities. For the same reason, we do not use WRK in our experiments on multi-tenant colocations in Section 5.4.

Given the number of experiments that we need to run, we created scripts to automate experimental runs (configuration, launch, and termination of domains and test workloads), data collection, power model building, and precision assessment. Each experimental run starts by setting the system in a clean and stable state. To deal with workloads' settling periods, we wait for 60 seconds between workload activations as well as deactivations.

5.2 Model Precision

Next, we evaluate the precision of the power models built with the proposed methodology. Depending on the workloads used to built the models, the same methodology can produce either *generic models* or *workload-specific models*. A generic model is built using micro-benchmarks such as those discussed in Section 4.1. Generic models are workload-agnostic and aim at making predictions on power consumption of workloads that have not been run before on the target machine, a feature that makes them rather attractive. By contrast, a workload-specific model, as its name suggests, is

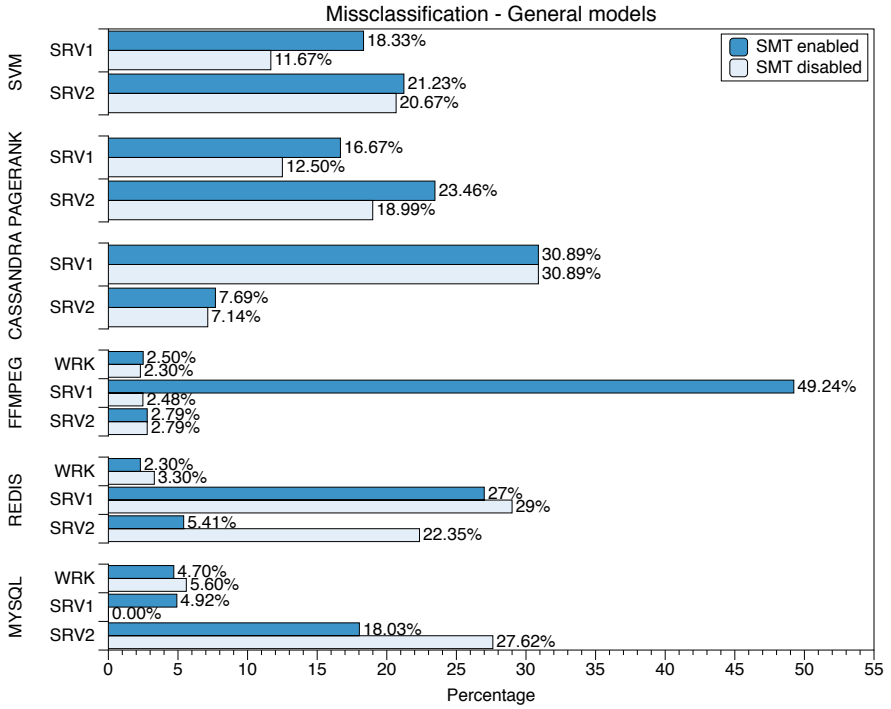


Fig. 6. Misclassification rates of generic models for the test workloads on the selected hardware platforms.

built with a particular workload to predict its power consumption; thus, such model includes a working regime classifier specialized for the target workload.

We expect in principle that, compared to generic models, workload-specific models have better precision as they offer lower *misclassification rates*[1] for the working regimes of the system when running the target workloads. However, their main drawback is that building a workload-specific model requires having the target workload in advance to collect its execution traces. Then, the key question we seek to answer here is *whether generic power models offer sufficiently high precision.*

We first report the misclassification rates of generic models for our test workloads on the selected hardware platforms, with SMT enabled and disabled. Figure 6 shows that in most cases SMT has no strong influence on misclassification rates. Two cases, however, exhibit large discrepancies in misclassification: (1) FFMPEG on SRV1 reports ~50% with SMT and only 2.48% without SMT; and (2) Redis on SRV2 reports 5.41% with SMT, but over 22% without SMT. These differences are due to design choices we made on the classifiers used in our evaluation. We deliberately use the same number of PMCs for every classifier for two reasons: first, to have a fair comparison between the different results, and second, to ensure that our methodology is practical in real-world scenarios, where the classifier's generation is the same across multiple machines. Note that such discrepancies can be easily mitigated by using additional PMCs as input features for improved classification performance, as discussed in Section 4.3.

Quite surprisingly, Figure 6 also shows that in some occasions classifiers of generic models may be able to perform virtually perfectly, like in the case of MySQL on SRV1 with SMT disabled. We presume that workloads with almost perfect classification are those that show better *affinity* with

[1] i.e., percentage of times the model's classifier incorrectly categorizes a working regime.

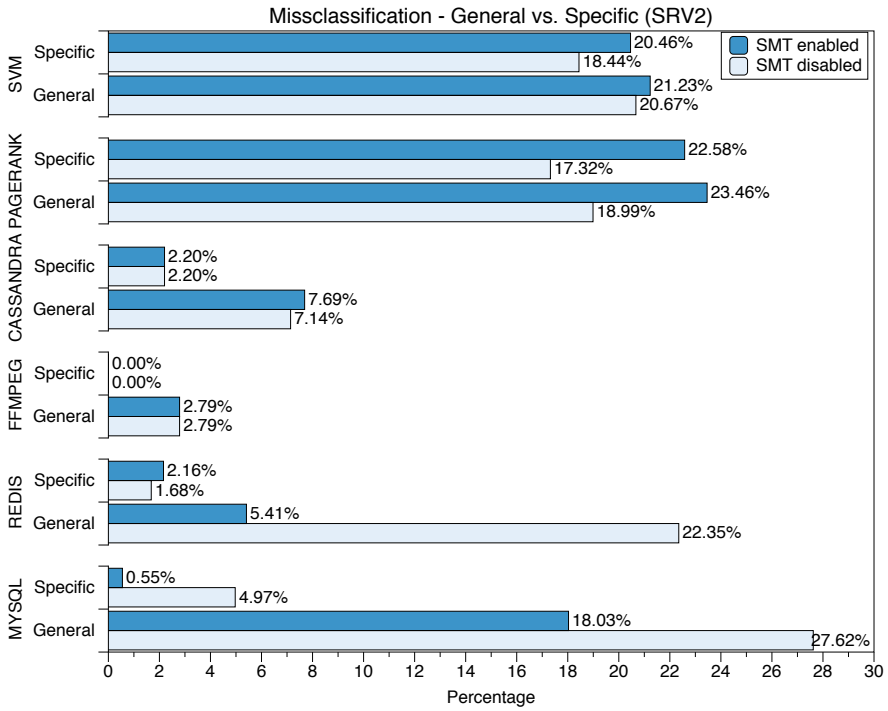


Fig. 7. Misclassification rates of generic and workload-specific models for the test workloads on SRV2.

the hardware platform they run on, meaning that they behave *as expected* on the machines from a workload-agnostic point of view.

Then, we compare the misclassification rates of generic and workload-specific models for the test workloads. Figure 7 only reports the results on SRV2 with SMT enabled and disabled, but similar results, omitted due to space limitations, were obtained on SRV1 and WRK. As expected, the misclassification rates of the workload-specific models are lower than those of the generic models across the test workloads with the same SMT setup; although in some cases (e.g., SVM and PageRank) they are comparable. It is worth noting that while FFMPEG reports a misclassification of ~50% with a generic model on SRV1 (see Figure 6), this time it reports excellent (and even perfect) classification with its specific model. The reason is that FFMPEG and other similar workloads usually work on a very stable and definite regime, which was not correctly identified during the training of the generic model's classifier. Hence, it is clear that the proposed methodology, combined with specifically targeted workloads, can produce workload-specific classifiers able to discriminate the relevant working regimes much better than the classifiers in generic models.

Finally, Figure 8 presents the estimation errors of generic models for our test workloads on the selected hardware platforms, with SMT enabled and disabled. The figure reports RMSE values (in Watts, as the bar length) and the mean relative errors (in percentage). Power measurements were obtained using Intel RAPL interface. The mean relative error is a metric commonly used that allows us to compare our results with those from the literature. It is calculated with respect to the range of the machine's *dynamic power*, defined as peak power minus base power.

Figure 8 shows that the generic power models exhibit good precision in all the cases considered. More precisely, we observe that:

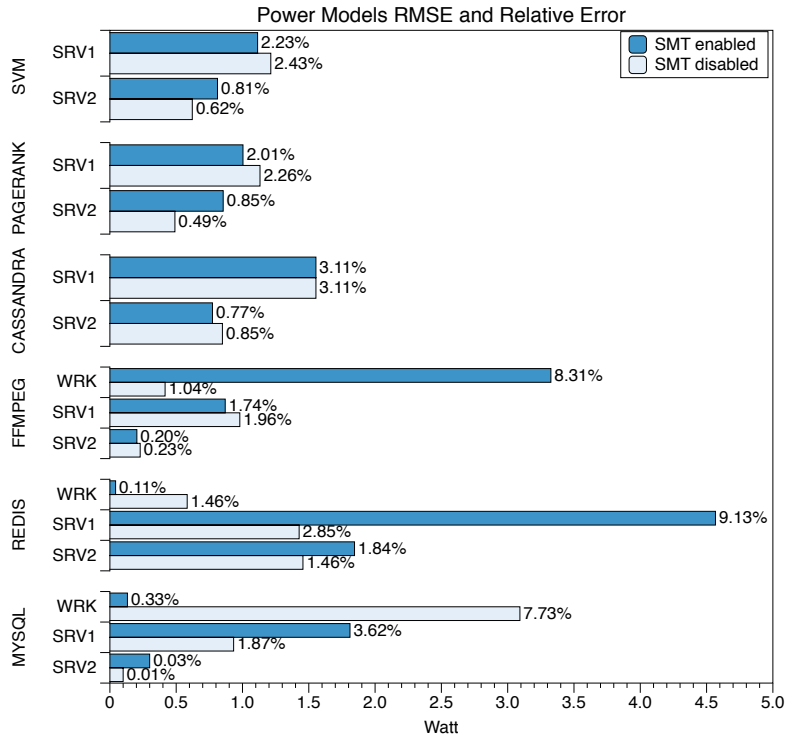


Fig. 8. RMSE and mean relative error (with respect to measured dynamic power consumption) of generic power models for the test workloads on the selected hardware platforms. RMSE (in Watts) is indicated by the length of the bars and the mean relative error (in percentage) is shown at the right side of the bars.

Table 4. Model portability for FFMPEG w.r.t. RMSE (in Watts).

Train / Test	WRK	SRV1	SRV2
WRK	3.32	0.85	—
SRV1	3.34	0.86	0.76
SRV2	—	0.98	0.20

Table 5. Model portability for Redis w.r.t. RMSE (in Watts).

Train / Test	WRK	SRV1	SRV2
WRK	0.58	1.42	—
SRV1	0.22	1.42	2.39
SRV2	—	9.33	1.45

Table 6. Model portability for MySQL w.r.t. RMSE (in Watts).

Train / Test	WRK	SRV1	SRV2
WRK	0.13	1.15	—
SRV1	17.83	1.81	17.15
SRV2	—	3.84	0.30

(1) RMSE is ~1W on average and less than 2W in most cases; only three cases report higher values, but do not exceed the reasonable limit of 5W.
(2) The mean relative error is ~2% on average, less than 4% in most cases, and in any case larger than 10%.

Therefore, our results indicate that generic power models can offer sufficiently high precision. Moreover, they outperform previous results in the literature [13, 15, 41].

5.3 Model Portability

As tenant migration is a common practice in cloud computing, model portability from one physical machine to a different one is a desirable feature. We are then interested in measuring the error of generic power models when they try to predict the power consumption of machines different from those the models were built for.

Here we focus on FFMPEG, Redis, and MySQL because, contrary to SVM, PageRank, and Cassandra, they were able to run on the three machines used for evaluation (WRK, SRV1, and SRV2). To

experiment with different system configurations, we run FFMPEG and MySQL with SMT enabled and Redis with SMT disabled. As before, power was measured with Intel RAPL interface.

Tables 4, 5 and 6 show the portability results of generic models for FFMPEG, Redis, and MySQL, respectively. In these tables, each row (except the top row) corresponds to a machine with a model built for it, and each column (except the leftmost column) corresponds to the machine on which the test workload actually runs. Each cell then contains the RMSE of a generic model applied to a test workload running on either the same machine for which the model was created or a different machine. RMSE values for the models' original target machines are on the main diagonals.

As expected, smaller errors tend to come from the machines which the models were built for. The reason is that during model creation the target platform is stressed (see Section 4.1) and the resulting model becomes somewhat bound to the underlying hardware. But more importantly, our results show that it is possible to use a generic power model built for a machine to predict the power consumption of a workload on a different machine, for which no power model exits. In this experiment, we observe only two cases of poor portability: SRV1's model with MySQL (Table 6), and SRV2's model with Redis (Table 5).

A key question is *how different the machines can be to still enable power model portability.* Not very different, our results only allow us to say, as the hardware platforms used in this work have certain similarities. For instance, WRK and SRV1 both have processors based on the same microarchitecture (Intel Sandy Bridge 2nd generation), and SRV1 and SRV2 have the same model of processors (Intel Xeon). In fact, it is clearly reasonable for similar hardware platforms to exhibit similar power behavior, and that is why we do not evaluate model portability between WRK and SRV2.

5.4 Comparing Colocations of Tenants

In this section, we investigate how generic power models can help us compare different colocations of tenants in terms of predicted power consumption. We refer to a *tenant colocation* as a specific set of tenants, each hosted in a separate domain, that run simultaneously on the same machine. A tenant colocation also stipulates the allocation of resources to its tenants (e.g., CPU count, amount of memory, and whether or not tenants share the same socket or physical cores).

Assume a colocation of $k > 1$ tenants. At runtime, series of hardware events can be collected and attributed to individual tenants, and using per-tenant event traces, a power model can estimate the power consumption $\hat{P}_I(j)$ of each colocated tenant j as if it was running *in isolation.* The total power consumption that would be required to run the same k tenants in isolation on separate machines can be calculated as $\hat{P}_I = \sum_{j=1}^{k} \hat{P}_I(j)$. Then, we estimate the relative improvement (RI) in power consumption of the tenant colocation with respect to running the tenants separately as:

$$RI = max(0, (\hat{P}_I - P_C)/P_C) \tag{5}$$

where P_C is the actual power consumption of the system with the colocated k tenants. Note that we disregard unprofitable colocations with negative RI values that may occur in practice. We use this metric to compare tenant colocations and identify those that promise better improvement in power consumption.

Next, we present a comparative study of different colocations with our test workloads using Equation (5). Our experiments run an increasing number of workloads until each workload cannot be assigned a separate physical core (i.e., two workloads never run on the same core). As before, each workload is hosted in a separate Xen domain. We evaluate colocations of *homogeneous* tenants by running multiple instances of individual workloads, as well as colocations of *heterogeneous* tenants by running workloads that exhibit distinct predominant behaviors.

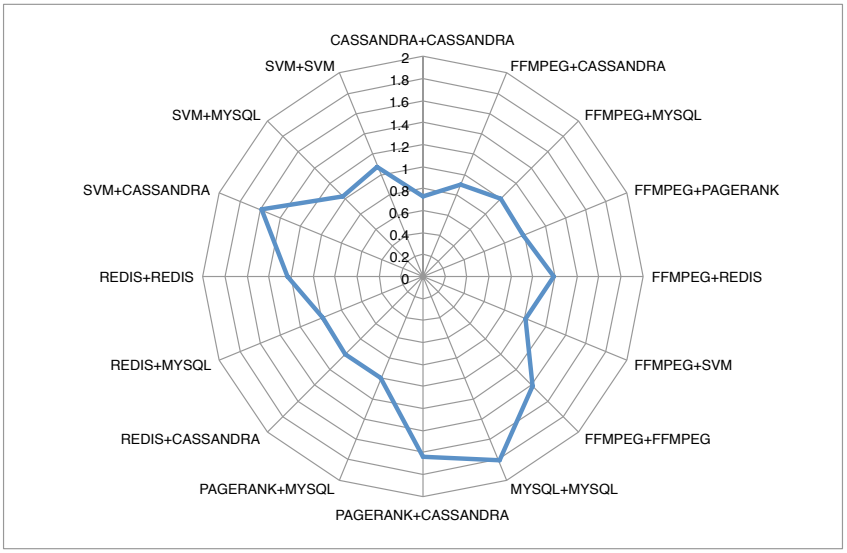


Fig. 9. Relative improvement in power consumption for different colocations of test workloads on SRV1, obtained using Equation (5).

We conduct our experiments only on the server machines SRV1 (with a 4-core processor) and SRV2 (with two 10-core processors) because, contrary to WRK, they are well equipped to simultaneously host multiple of our test workloads. SMT is enabled in both machines. On SRV2, we distribute workloads uniformly across both processors, whenever possible. Moreover, we measure each machine's power consumption with a *WattsUp* meter since we are interested in the system-wide power consumption.

Figure 9 shows the relative improvement (*RI*) in power consumption for different colocations of test workloads on SRV1. Due to SRV1's hardware limitation, we only run two workloads at a time. We observe that most colocations report *RI* values around 1.0; this means that these colocations could roughly save the power consumption of a dedicated machine running one of the workloads. We can also see that `MYSQL+MYSQL`, `PAGERANK+CASSANDRA`, and `SVM+CASSANDRA` offer better improvements. For colocations like `PAGERANK+CASSANDRA` and `SVM+CASSANDRA` this is understandable because the workloads have orthogonal needs – one is mostly CPU-bound while the other is mostly memory- and I/O-bound – and together make a balanced use of resources, splitting the fixed cost of base power. The case of `MYSQL+MYSQL` is, on the other hand, rather surprising and requires a closer look for us to provide an explanation.

We also evaluate the *RI* metric for a number of colocations on SRV2. We confine this experiment to using only one of the two processors in the machine. The results are shown in Figure 10. This time we are able to run a larger number of simultaneous workloads because each processor in SRV2 is more powerful than SRV1's processor. We observe that most colocations report $RI \geq 2$, which suggests that SRV2 may be more favorable than SRV1 to multi-tenant consolidation in terms of power consumption. Moreover, it is worth noting that `SVM+CASSANDRA`, which is one of the best colocations on SRV1 with $RI \approx 1.6$ (Figure 9), turns out to be one of the worst on SRV2 with $RI \approx 1.0$. Although the difference in *RI* is not really significant, it illustrates a case in which the *RI* metric could guide the decisions on colocation of tenants on different machines (e.g., run `SVM+CASSANDRA` side by side on SRV1, but not on SRV2).

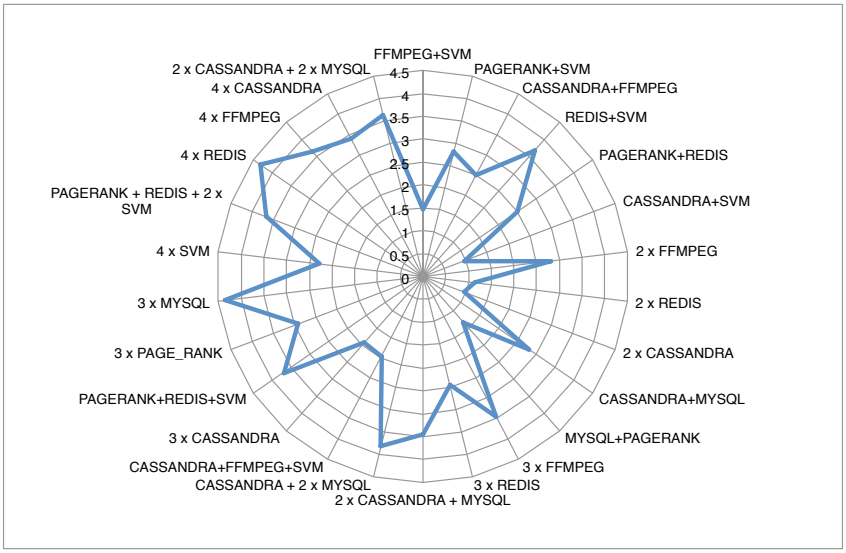


Fig. 10. Relative improvement in power consumption for different colocations of test workloads on SRV2's single processor, obtained using Equation (5).

Our last experiment uses SRV2's two processors. Due to the large number of colocations, we divide our results into two figures. Figure 11 reports the results with *homogeneous* workloads and Figure 12 with *heterogeneous* workloads.

Figure 11 shows that in almost all cases *RI* increases with the number of colocated homogeneous workloads, as expected. But, the way *RI* rises is workload-specific (e.g., three or more FFMPEG instances offer better power improvements than the same number of CASSANDRA instances). Regarding heterogeneous colocations, Figure 12[2] indicates that a good balance of homogeneous workloads in a heterogeneous colocation is key to achieve large relative improvements in power consumption. Consider the case of 2C+3F+M and C+4F+M; both colocations include six workloads of three different types, but adding a FFMPEG instance instead of a CASSANDRA instance leads to an increase in *RI* from 6 to 8. By contrast, another six-tenant colocation 3F+P+R+S only reports $RI \approx 5.5$, highlighting again the importance of choosing the right set of colocated workloads. In addition, a comparison between Figure 11 and Figure 12 reveals that workload heterogeneity is key for tenant colocations to reach high *RI* values (e.g., while 6xFFMPEG only reports $RI \approx 6$, C+4F+M, also with six workloads, reports $RI \approx 8$).

In this section, we have shown that *RI*, given by Equation (5), can serve as a discriminator for power profiles of tenant colocations. A cluster scheduler could use *RI*, along with other performance metrics, to decide which tenants should reside together in the different nodes in order to reduce the cluster's power consumption. By relying on generic power models and *RI*, the scheduler may just treat tenants as "black boxes".

6 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a power modeling methodology that tackles the complexity introduced by workload heterogeneity through working regime identification. The idea is to identify the system's working states and build a power model around them. We showed that

[2] Note that the names of the workloads have been replaced by their initial letter to improve the chart's readability.

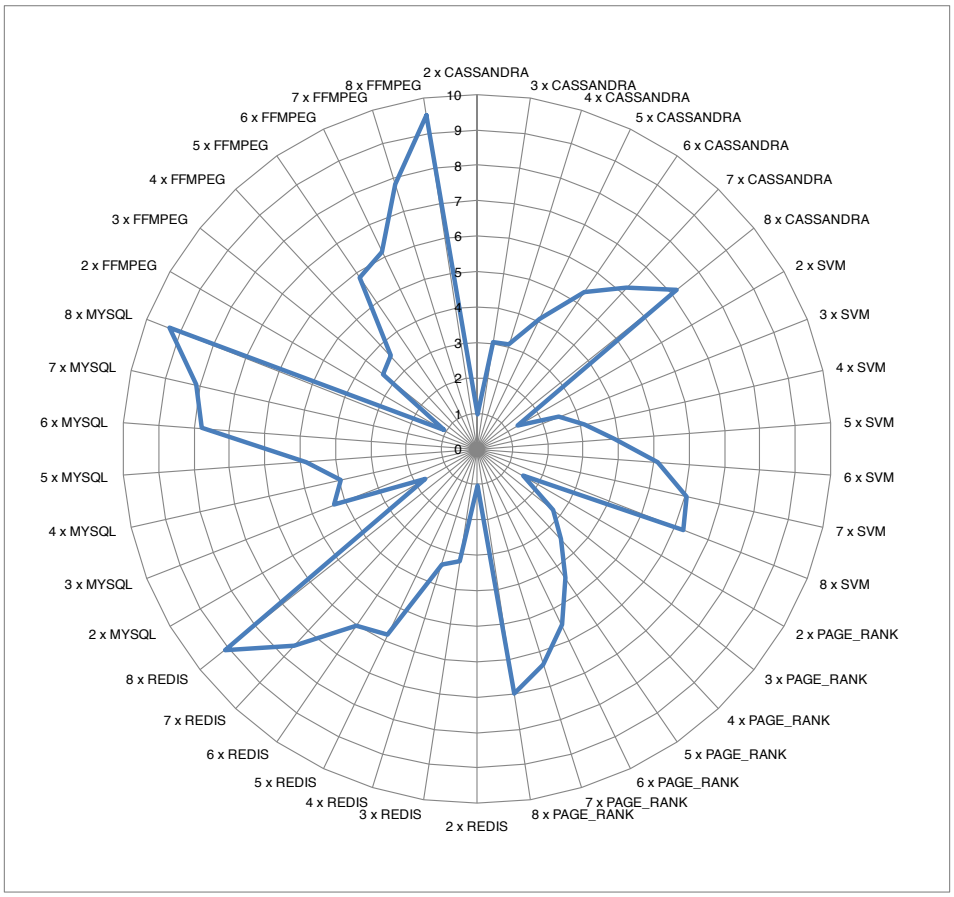


Fig. 11. Relative improvement in power consumption for different colocations of homogeneous workloads on SRV2's two processors, obtained using Equation (5).

such an approach is required as the heterogeneity of tenants strongly influence system power consumption. Our methodology was able to outperform previous results in the literature, even in the worst cases, using workload-agnostic (generic) and workload-specific power models.

In addition, we established that it is possible for generic power models to predict the impact of migrating a tenant to a different hardware platform. We also showed how the models can help a cluster scheduler compare colocations of tenants, leading to more power efficient workload consolidations across server farms.

This work focused on power modeling, but to thoroughly explore the topic we should also consider workload performance. In the future, we plan to investigate effective performance-power policies and extend our current results with performance analyses. Another desirable extension is related to the benchmarks selected to build generic models; in fact, some large modeling errors stem from still unexplored hardware features in the machines. Finally, an important aspect is the evaluation of numerous tenant colocations. The number of possible colocations grows rapidly with the tenant count. We plan to extend our methodology to aggressively discard unfavorable colocations of tenants.

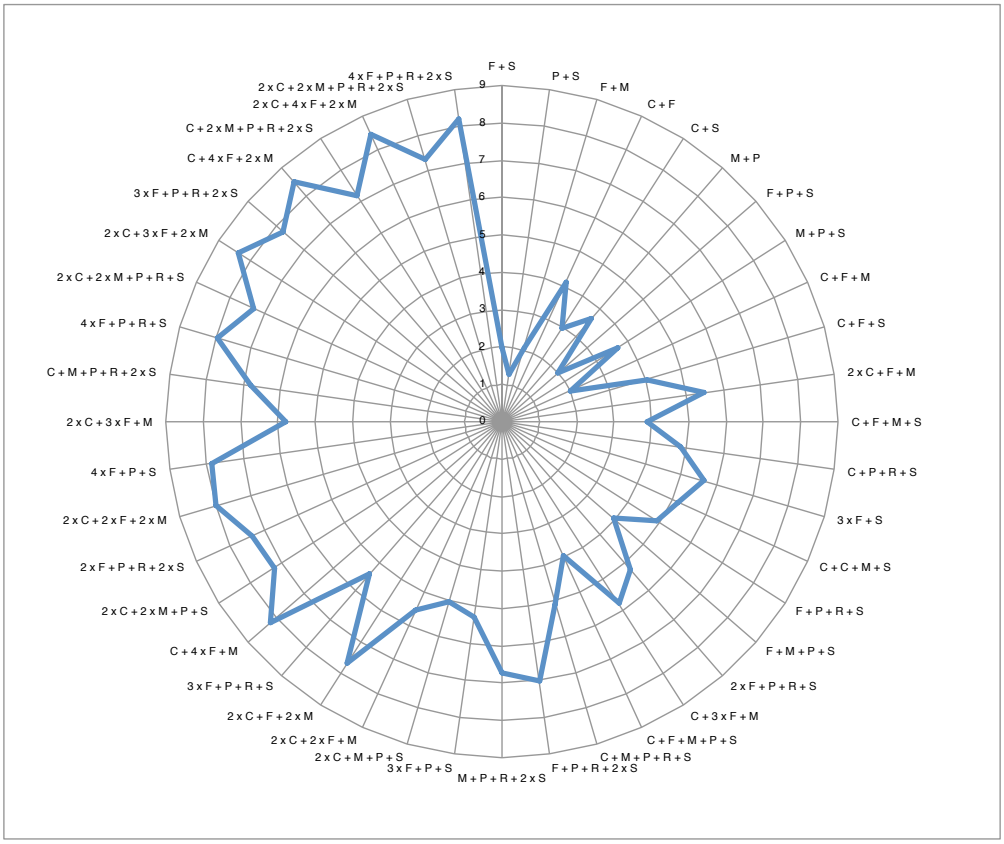


Fig. 12. Relative improvement in power consumption for different colocations of heterogeneous workloads on SRV2's two processors, obtained using Equation (5).

REFERENCES

[1] Apache Cassandra. http://cassandra.apache.org/. Accessed: 2017-06-09.

[2] Benchmark suite for Apache Spark. https://github.com/SparkTC/spark-bench. Accessed: 2017-06-09.

[3] Ffmpeg. https://ffmpeg.org. Accessed: 2017-06-09.

[4] MySQL benchmark tool. https://dev.mysql.com/downloads/benchmarks.html. Accessed: 2017-06-09.

[5] Redis. http://redis.io/topics/benchmarks. Accessed: 2017-06-09.

[6] Watts up? plug load meters. https://www.wattsupmeters.com/secure/products.php?pn=0. Accessed: 2017-06-07.

[7] IOzone filesystem benchmark. http://www.iozone.org, 2007.

[8] Acun, B., Miller, P., and Kale, L. V. Variation among processors under turbo boost in HPC systems. In *the 2016 International Conference on Supercomputing* (2016), pp. 6:1–6:12.

[9] Ali, I., and Meghanathan, N. Virtual machines and networks-installation, performance study, advantages and virtualization options. *arXiv preprint arXiv:1105.0061* (2011).

[10] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., et al. Above the clouds: A Berkeley view of cloud computing. Tech. rep., 2009.

[11] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *the 19th ACM Symposium on Operating Systems Principles* (2003), pp. 164–177.

[12] Bellosa, F. The benefits of event-driven energy accounting in power-sensitive systems. In *the 9th ACM SIGOPS European Workshop. Beyond the PC: New challenges for the operating system* (2000), pp. 37–42.

[13] Beloglazov, A., Buyya, R., Lee, Y. C., Zomaya, A., et al. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers 82*, 2 (2011), 47–111.

[14] Bertran, R., Becerra, Y., Carrera, D., Beltran, V., Gonzàlez, 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), 457–468.

[15] Bircher, W. L., and John, L. K. Complete system power estimation: A trickle-down approach based on performance events. In *the IEEE International Symposium on Performance Analysis of Systems & Software* (2007), pp. 158–168.

[16] Bircher, W. L., Valluri, M., Law, J., and John, L. K. Runtime identification of microprocessor energy saving opportunities. In *the 2005 Int'l Symposium on Low Power Electronics and Design* (2005), pp. 275–280.

[17] Dash, M., and Liu, H. Feature selection for classification. *Intelligent data analysis 1*, 3 (1997), 131–156.

[18] Delimitrou, C., and Kozyrakis, C. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *the 18th ACM Int'l Conference on Architectural Support for Programming Languages and Operating Systems* (2013), pp. 77–88.

[19] Delimitrou, C., and Kozyrakis, C. Quasar: Resource-efficient and QoS-aware cluster management. In *the 19th ACM Int'l Conference on Architectural Support for Programming Languages and Operating Systems* (2014), pp. 127–144.

[20] Dhiman, G., Mihic, K., and Rosing, T. A system for online power prediction in virtualized environments using gaussian mixture models. In *the 47th ACM/IEEE Design Automation Conference* (2010), pp. 807–812.

[21] Division, N. A. S. Nas parallel benchmarks (npb). http://www.nas.nasa.gov/publications/npb.html, 2016.

[22] Fan, X., Weber, W.-D., and Barroso, L. A. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News* (2007), vol. 35, pp. 13–23.

[23] Ferroni, M., Colmenares, J. A., Hofmeyr, S., Kubiatowicz, J. D., and Santambrogio, M. D. Enabling power-awareness for the Xen hypervisor. In *the 2016 Embedded Operating System Workshop* (October 2016).

[24] Ferroni, M., Corna, A., Damiani, A., Brondolin, R., Kubiatowicz, J. D., Sciuto, D., and Santambrogio, M. D. Marc: a resource consumption modelling service for self-aware autonomous agents. *Transaction on Autonomous and Adaptive Systems* (Accepted to appear).

[25] Gu, C., Shi, P., Shi, S., Huang, H., and Jia, X. A tree regression-based approach for vm power metering. *IEEE Access 3* (2015), 610–621.

[26] Heiser, G. The role of virtualization in embedded systems. In *the 1st Workshop on Isolation and Integration in Embedded Systems* (2008), ACM, pp. 11–16.

[27] Isci, C., and Martonosi, M. Runtime power monitoring in high-end processors: Methodology and empirical data. In *the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (2003), p. 93.

[28] Kansal, A., Zhao, F., Liu, J., Kothari, N., and Bhattacharya, A. A. Virtual machine power metering and provisioning. In *the 1st ACM Symposium on Cloud computing* (2010), ACM, pp. 39–50.

[29] Kononenko, I. Estimating attributes: analysis and extensions of relief. In *Machine Learning: ECML-94* (1994), Springer, pp. 171–182.

[30] Kumar, R., and Charu, S. Comparison between cloud computing, grid computing, cluster computing and virtualization. *International Journal of Modern Computer Science and Applications (IJMCSA) 3*, 1 (January 2015).

[31] Lee, K.-J., and Skadron, K. Using performance counters for runtime temperature sensing in high-performance processors. In *the 19th IEEE International Parallel and Distributed Processing Symposium* (2005).

[32] Li, T., and John, L. K. Run-time modeling and estimation of operating system power consumption. In *ACM SIGMETRICS Performance Evaluation Review* (2003), vol. 31, ACM, pp. 160–171.

[33] Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., and Kozyrakis, C. Heracles: improving resource efficiency at scale. In *ACM SIGARCH Computer Architecture News* (2015), vol. 43, pp. 450–462.

[34] Mergen, M. F., Uhlig, V., Krieger, O., and Xenidis, J. Virtualization for high-performance computing. *ACM SIGOPS Operating Systems Review 40*, 2 (2006), 8–11.

[35] Mucci, P. J. Cachebench. http://www.weblearn.hs-bremen.de/risse/RST/WS06/x86_SUN/Sourcen/LLCBench/www/cachebench.html, 2007.

[36] Rosenblatt, M., et al. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics 27*, 3 (1956), 832–837.

[37] Rotem, E., Naveh, A., Ananthakrishnan, A., Weissmann, E., and Rajwan, D. Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *IEEE Micro 32*, 2 (Mar. 2012), 20–27.

[38] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. Omega: flexible, scalable schedulers for large compute clusters. In *the 8th ACM European Conference on Computer Systems* (2013), pp. 351–364.

[39] Semnanian, A. A., Pham, J., Englert, B., and Wu, X. Virtualization technology and its impact on computer hardware architecture. In *the 8th Int'l Conference on Information Technology: New Generations* (2011), pp. 719–724.

[40] Takouna, I., Dawoud, W., and Meinel, C. Accurate multicore processor power models for power-aware resource management. In *the 2011 IEEE 9th Int'l Conference on Dependable, Autonomic and Secure Computing* (2011), pp. 419–426.

[41] Yang, H., Zhao, Q., Luan, Z., and Qian, D. iMeter: An integrated vm power model based on performance profiling. *Future Generation Computer Systems 36* (2014), 267–286.