# Predicting Server Power Consumption from Standard Rating Results

Jóakim von Kistowski
University of Würzburg
Germany
joakim.kistowski@uni-wuerzburg.de

Johannes Grohmann
University of Würzburg
Germany
johannes.grohmann@uni-wuerzburg.de

Norbert Schmitt
University of Würzburg
Germany
norbert.schmitt@uni-wuerzburg.de

Samuel Kounev
University of Würzburg
Germany
samuel.kounev@uni-wuerzburg.de

## ABSTRACT

Data center providers and server operators try to reduce the power consumption of their servers. Finding an energy efficient server for a specific target application is a first step in this regard. Estimating the power consumption of an application on an unavailable server is difficult, as nameplate power values are generally overestimations. Offline power models are able to predict the consumption accurately, but are usually intended for system design, requiring very specific and detailed knowledge about the system under consideration.

In this paper, we introduce an offline power prediction method that uses the results of standard power rating tools. The method predicts the power consumption of a specific application for multiple load levels on a target server that is otherwise unavailable for testing. We evaluate our approach by predicting the power consumption of three applications on different physical servers. Our method is able to achieve an average prediction error of 9.49% for three workloads running on real-world, physical servers.

## CCS CONCEPTS

• **Hardware → Energy metering**; **Platform power issues**; **Enterprise level and data centers power issues**; • **Software and its engineering** → *Software performance*;

## KEYWORDS

Power, Energy Efficiency, Performance, Prediction, Interpolation, Regression, Benchmarking, Load Level

**ACM Reference Format:**
Jóakim von Kistowski, Johannes Grohmann, Norbert Schmitt, and Samuel Kounev. 2019. Predicting Server Power Consumption from Standard Rating Results. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3310298

## 1 INTRODUCTION

The energy efficiency of servers has become a significant issue as data center energy consumption has risen dramatically over the past decade. In 2010, the U.S. Environmental Protection Agency (U.S. EPA) estimated 3% of all electricity consumed in the U.S. to be used in running data centers [24]. According to a New York Times study from 2012, data centers worldwide consume about 30 billion watts per hour [2]. This leads to high operational costs for data center and server operators. Consequently, data center provisioners would ideally attempt to purchase servers which are energy efficient regarding their intended use.

When purchasing servers, their energy efficiency and power consumption can be estimated using nameplate information and standard benchmark results. However, nameplate values are usually overestimated and inaccurate and standard benchmarks do not reflect the target application to be run on the servers in question. Due to this, server providers rely on educated guesses and experience when trying to choose the most efficient and least consuming servers for their specific application. The ability to accurately assess the power consumption of the servers under consideration for their intended application would enable planning and provisioning of energy efficient server landscapes.

Existing power models are either very generic and do not consider workload, or they are assumed to be trained at run-time. Models that predict power for unavailable systems and components, such as [9] and [22], are primarily intended for server and hardware component design. They require very detailed knowledge on system internals, which would be available to a system designer, but not necessarily to someone intending to buy such a system.

This paper proposes a prediction method for power consumption of servers based on standard rating tool results. The prediction method uses publicly available rating data to predict the power consumption of servers under consideration for a target application. Specifically, the method uses data available from the SPEC Server Efficiency Rating Tool (SERT), which is required to run for U.S. Environmental Protection Agency (E.P.A.) Energy Star labeling. Using our method, a server provisioner can predict the power consumption of a future server running a target application even without access to it. Of course, this also implies that the server provisioner does not have to perform any measurements on the server under consideration.

Predicting the power consumption of servers using standard rating tool results is difficult, as this result set is relatively small. To address this issue, we consider multiple prediction formalisms and introduce a parameter optimization method. We also investigate the use of interpolation to generate additional data points for training and prediction.

The goal of this paper is to provide accurate prediction of server power consumption for a target workload at multiple application load levels, based only on the SERT results of the target server. We envision the result of this prediction, containing information on the concrete workload under consideration and multiple load levels, to be of use for planners and potential server customers when making their decisions. In a nutshell, the core contributions of the paper are:

(1) We introduce an offline prediction method for server power consumption based on standard rating tool results.
(2) We present a parameter optimization method for automated tuning of prediction formalisms.
(3) We investigate and show the use of interpolation methods to create additional training and prediction data for the power prediction domain.

We evaluate our offline prediction method by prediction the power consumption of three target workloads on separate physical servers. We evaluate the prediction accuracy the effects of parameter optimization and interpolation. We show that our method can predict the power consumption of applications for an unavailable server using only standard rating tool results with a mean average absolute error of 9.49%, measured using real-world, physical servers and workloads. We also show that our combination of interpolation and parameter optimization methods greatly aids in achieving accurate predictions in a domain with little training and prediction data.

The remainder of this paper is structured as follows: Section 2 describes the related power models and prediction mechanisms. Section 3 introduces the SERT rating tool, and workloads, and their relevance for our power prediction. We then describe our offline power prediction approach in detail in Section 4. Next, we evaluate our prediction accuracy in Section 5. Finally, the paper concludes in Section 6.

## 2 RELATED WORK

We group related work in three non-exclusive areas: We examine power models in the two non-exclusive categories of *general offline power models*, *generic server power models* (*hardware-based* and *workload-based*), models for *energy-aware power management*. We discuss the power benchmarks and workloads for our workload-based model separately in Section 3.

**Offline power models**, such as [4], [22], and [18] are intended for use at design time. Such models can be architecture-based, such as [4] and [37], which expand upon software architecture models for design time power comparisons. These models are intended to compare software design alternatives for distributed systems and focus on the relative power prediction results with reduced absolute accuracy of single server predictions. In contrast, offline power models for hardware designers focus on maximum accuracy for single hardware components by requiring extremely detailed modeling of their properties. These models, such as the CPU model of [9] and [22], are usually intended for hardware system designers, analyzing their potential device architectures. Modeling approaches of such a granularity are also employed on a full server level in [18], which allows for detailed modeling of full systems. In contrast to these models, our model learns purely from standard benchmark and application efficiency measurement, requiring no explicit modeling by the user.

Generic **server power models** characterize or predict the power consumption of a single physical server. They have some overlap with offline power models (e.g., [18]), yet many are not specifically designed for offline prediction. [32] provides an overview of generalized, generic full-system power models. These models can utilize a variety of methods, such as interpolation [16, 42], regression [26, 27], or others [14]. They also vary regarding their purpose and, especially, regarding the type of data that they use to model the power consumption. For this work, we distinguish between hardware-based and workload-based power models.

**Hardware-based models**, such as [26, 27] and [14] are based on system-level data, such as architectural parameters, performance counters [6, 11, 12, 20, 29, 34, 35, 39] and utilization metrics. This data is often collected at run-time or during a calibration step from the system under consideration. Hardware-based models also are often highly specialized to the calibration hardware. Further complicating hardware-based models is the necessary expert knowledge to build them. The Intel®manual [19] lists over 300 performance counters for the HaswellEP microarchitecture and a performance counter available on one platform must not necessarily available on another. Thus making hardware-based models often less portable than an offline power model that can be constructed or trained from less hardware specific data.

**Workload-based models**, such as [42] and [15], are usually trained for specific workloads and use application-level parameters such as request arrival rates. The model of this paper can also be counted in this class. However, with the exception of [43], none of the existing workload-based models use standard rating results to enable a black-box power prediction for otherwise unavailable servers. Only [43] is of special note in this regard, as it also predicts power using the SPEC SERT. However, its use-case is quite different, as it predicts the power overhead of VM hypervisors.

**Energy-aware power management** techniques may employ generic power models or specific online power prediction models, such as [41], to predict the power consumption of the system states that might result from management decisions. The typical goal is to minimize power consumption within certain Quality-of-Service (QoS) constraints [5], or to maximize overall efficiency [21]. The common technical mechanism by which management systems achieve this is VM migration [5, 21, 38, 40]. The impact of the power management decisions is often estimated using established basic power prediction methods. For example, [5] use a utilization-based linear power model described by [32], whereas [40] use a quadratic power model. Other works construct their own model of the underlying systems, such as [38], which model a server farm using stochastic Petri nets. Our proposed model differs from this class of models, as it is primarily intended for offline prediction at a time when the system under consideration is not yet available to the person making the prediction.

## 3 SERT

The SPEC Server Efficiency Rating Tool (SERT) has been developed by the SPEC OSG Power Committee as a tool for the analysis and evaluation of the energy efficiency of server systems. Its design and development goals and process have been introduced in [24]. In contrast to energy-efficiency benchmarks, such as JouleSort [33], SPECpower_ssj2008 [23], and the TPC-Energy benchmarks [31], SERT does not execute an application from a specific domain. It does not aim to emulate real world end user workloads, but instead provides a set of focused synthetic micro-workloads called worklets that exercise selected aspects of the Server (or System) Under Test (SUT). The worklets have been developed to exercise the processor, memory, and storage I/O subsystems.

For each of the server components to be stressed, SERT offers a range of worklets designed to exercise the targeted component in a different manner. This allows for thorough analysis of system energy behavior under different workload types designed to target the same component. As an example, the CryptoAES worklet profits from both specialized instruction sets, as well as better CPU to memory connectivity, whereas the SOR worklet primarily scales with processor frequency.

According to [3], servers nowadays spend most of their time in a CPU utilization range between 10% and 50%. As a result, SERT and its worklets are designed for the measurement of system energy efficiency at multiple load levels. This sets it further apart from conventional performance benchmarks, such as SPEC CPU [10], which targets maximum load and performance. To achieve workload execution at different load levels, SERT calibrates the load by determining the maximum transaction rate for the given worklet on the SUT. The maximum transaction rate is measured by running as many of the worklet's transactions as possible concurrently on each client (i.e. utilizing all logical cores and not introducing artificial waiting delays). This calibrated rate is then set as the 100% load level for all consecutive runs. For each target load level (e.g., 100%, 75%, 50%, 25%), SERT calculates the target transaction rate and derives the corresponding mean time from the start of one transaction to the start of the next transaction. During the measurement interval, these delays are randomized using an exponential distribution that statistically converges to the desired transaction rate. As a result, lower target loads consist of short bursts of activity separated by periods of inactivity. The separate load levels per worklet are useful when using SERT results to train power models, as it provides multiple independent measurement results for each worklet that can be used for model training. Note, that the load levels are throughput based and do not indicate CPU utilization (this is a common misconception).

### 3.1 Worklets

In version 2.0, SERT features seven CPU worklets. Each worklet is measured at four load levels (25%, 50%, 75%, 100%), with the exception of SSJ:

(1) **Compress**: De-/compresses data using a modified Lempel-Ziv-Welch (LZW) method [44]
(2) **CryptoAES**: Encrypts/decrypts data using the AES or DES block cipher algorithms
(3) **LU**: Computes the LU factorization of a dense matrix using partial pivoting
(4) **SHA256**: Performs SHA-256 hashing transformations on a byte array
(5) **SOR** (Jacobi Successive Over-Relaxation): Exercises typical access patterns in finite difference applications
(6) **SORT** Sorts a randomized 64-bit integer array during each transaction
(7) **Hybrid / SSJ**: The hybrid SSJ worklet stresses both CPU and memory, with either serving as the primary bottleneck, depending on system configuration. SSJ Performs multiple different simultaneous transactions, simulating an enterprise application. SSJ is measured at eight load levels, instead of the four levels of the other CPU worklets

The two storage worklets, described in [25] are run at two load levels (50%, and 100%). The **Random** and **Sequential** worklets perform random and sequential read/write operations on the SUT's internal storage.

In version 2.0, SERT features seven CPU worklets. Each worklet is measured at four load levels (25%, 50%, 75%, 100%), with the exception of SSJ: *Compress*, *CryptoAES*, *LU*, *SHA256*, *SOR*, *SORT*, and *SSJ*. SSJ is measured at eight load levels, instead of the four levels of the other CPU worklets. In addition to the CPU worklets, the two storage worklets are run at two load levels (50%, and 100%).

Finally, SERT features two memory worklets: *Flood* and *Capacity*. Flood tests the SUT's memory bandwidth, whereas Capacity tests its capacity. However, the memory worklets do not use the same load level scaling mechanism as the other worklets. Each features two load levels that differ in the amount of memory reserved for the worklet, instead of scaling with the transaction rate. Because of this, we expect these worklets not be as useful in training power models as the storage and CPU worklets.

The major challenge in training models based on SERT results is twofold: *1)* The amount of data points per worklet is relatively small and *2)* it varies between the worklets. The load levels can be configured in theory, but standard compliant runs use the load levels described in this section. As we intend our model to be used with publicly available results, we must deal with the small and varying load level numbers.

## 4 OFFLINE POWER PREDICTION

The goal of our offline power prediction approach is the prediction of the power consumed by a *target application* running on a *target server* not yet available to the operator currently running the application. The general idea behind the prediction approach is as follows: The operator has a *current server* on which the *target application* in question is being executed or upon which it can be deployed for testing. The operator measures the performance and power consumption of the target application for multiple throughput load levels on this server. The prediction also requires a SERT result for the current server, which can be measured or obtained using a public database. Finally, it also requires a SERT result for the *target* server, which can be obtained from the vendor or a public database.

Predicting the power consumption of the target application for multiple potential target devices can help decision makers when

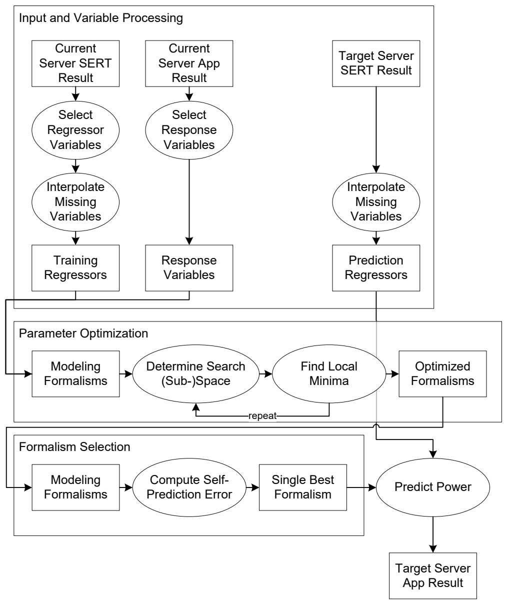
Input and Variable Processing
Current Server SERT Result
Current Server App Result
Target Server SERT Result
Select Regressor Variables
Select Response Variables
Interpolate Missing Variables
Interpolate Missing Variables
Training Regressors
Response Variables
Prediction Regressors
Parameter Optimization
Modeling Formalisms
Determine Search (Sub-)Space
Find Local Minima
Optimized Formalisms
repeat
Formalism Selection
Modeling Formalisms
Compute Self-Prediction Error
Single Best Formalism
Predict Power
Target Server App Result

**Figure 1: Outline of Power Prediction Approach**

deciding with which device to provision their cluster or data center. In addition, this approach could also be used in a to simulate general software placement context. An outline of the overall approach is illustrated in Figure 1. The approach is modular, allowing the use of a regression method from a pool of methods. Specifically we consider the following methods, where each method can be fine-tuned using the parameters exposed by their implementation (we use the Smile [28] library for Java):

- *Regression Tree (CART)*: the maximum number of leaf nodes in a tree (`max nodes`) and the number of instances in a node before splitting (`node size`)
- *Random Forest*: the number of trees in the forest (`num trees`)
- *Gradient Tree Boost*: the number of trees in the forest (`num trees`)
- *Gaussian Process Regression*: the `shrinkage` regularization parameter of the gaussian processess's kernel and the `kernel width`

## 4.1 Regressor and Response Variables

In general, power prediction is a prediction problem on a continuous scale and can thus be posed as a regression problem. These problem statements can be solved by various regression and/or classification algorithms. In general, regression problems have the following form (Eq. 1):

$$Y \approx f(\boldsymbol{X}, \boldsymbol{\beta}) \tag{1}$$

where $\boldsymbol{Y}$ is the vector of *response* variables (also called *dependent* variables), $\boldsymbol{X}$ the set of *regressor* variables (also called *independent* variables), and $\boldsymbol{\beta}$ the set of regression *parameters* to be trained. In this work, we map our domain specific measurement results to generic regressor variables and and response variables, allowing for a range of regression and classification models to be used, as opposed to limiting ourselves to a single model, such as e.g., linear regression. Note that not all of our models train a single regression parameter vector $\beta$. However, they are all capable of proceesing our set of regressor and response variables. For this reason, we make no assumptions on how or if the prediction model models $\beta$.

### 4.1.1 Target Application Power Response Variable for Training.

We measure the power consumption of the target application at multiple load levels on the *current server*. We then construct the training response vector from these power consumption results. As an example, the measurements at four load levels (100%, 75%, 50%, 25%), the training response vector $y$ would be constructed as shown in Equation 2:

$$\boldsymbol{y} = \begin{bmatrix} power(App_{100\%}) \\ power(App_{75\%}) \\ power(App_{50\%}) \\ power(App_{25\%}) \end{bmatrix}. \tag{2}$$

Note that the number of load levels may have great impact on the prediction accuracy. Specifically, not all load levels measured for the target application may have measurement counterparts for all worklets in the SERT (and thus in our set $X$ of independent regressor variables). We tackle this issue using interpolation in Section 4.3. As explained in Section 3, the load levels are derived from the workload's throughput. They describe the current throughput in relation to the maximum throughput achievable on the SUT. Consequently, this approach is applicable to any application with a measurable throughput. For many applications, this would be a request rate (e.g., HTTP requests per second for web applications), where 100% load would be the maximum request rate the SUT could handle.

### 4.1.2 SERT Results as Regressor Variables.

The matrix $X$ of independent regressor variables is constructed from the per-load level measurement results of the separate worklets. Specifically, we construct a vector from a single measurement metric, such as power or performance over the load levels of a worklet. We consider the following metrics for construction of our vectors:

- **Performance**: The average throughput of the worklet at the specified load level (in $s^{-1}$).
- **Power Consumption**: The average power consumption of the SUT when running the worklet at the specified load level (in $W$).

We do not consider the measured temperature, as it is measured as a control metric at the SUT inlet and thus independent from the system state. Equation 3 shows an example regressor variable matrix $X$ that uses the load level percentages and power consumption of several worklets with four load levels.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0.75 & 0.5 & 0.25 \\ pwr(Com._{100\%}) & \multicolumn{2}{c}{\cdots\cdots\cdots} & pwr(Com._{25\%}) \\ pwr(AES_{100\%}) & \multicolumn{2}{c}{\cdots\cdots\cdots} & pwr(AES_{25\%}) \\ \vdots & \vdots & \vdots & \vdots \\ pwr(SSJ_{100\%}) & \multicolumn{2}{c}{\cdots\cdots\cdots} & pwr(SSJ_{25\%}) \end{bmatrix}. \tag{3}$$

Again, the number of measured load levels is important and affects accuracy. In addition, the worklets within SERT are measured with different load level counts. However, many regression methods require training vectors of equal size. In our case, this would imply measurements with the same load level counts for all worklets, which is not the case based on the measurement data alone. To generate this equal load level count and create training vectors of equal size, our feature engineering must either remove load levels or create new ones. Specifically, we address this issue threefold by *1)* discarding load levels, *2)* discarding worklets, and *3)* interpolation (see Section 4.3). First, we optionally discard load levels that have no equivalent load level in the prediction (e.g., *SSJ* is measured with more load levels than any other worklet). Secondly, we discard some worklets that do not fit into the load level schema (such as *Capacity*) or feature too few load levels for accurate interpolation (*Flood* and the storage worklets). Finally, we apply interpolation for worklets with few load level measurements (see Section 4.3). In addition, we can add the power consumption of the *Idle* worklet as the power consumption of each worklet at 0% load. We expect the success of each of those measures to vary depending on the prediction method used.

*4.1.3 System Power as Expected Output.* The models predict system power consumption for each of the load levels of the application under consideration running on the target server. The number of load levels is implicitly specified by the size of the training response vector $y$.

## 4.2 Prediction Formalisms under Consideration

We use four underlying prediction formalisms for evaulation of the power prediction: three variations of regression trees and Gaussian mixture models. These formalisms are embedded in the enclosing power prediction framework. However, the framework is modularized and therefore supports any generic regression algorithm.

As tree algorithms, we use Regression Trees (CART) [8], Gradient Tree Boosting [17], and Random Forests [7]. Regression Trees are built using binary recursive partitioning, i.e., repeated iterative splitting of data into partitions or branches. Using CART, a tree is grown from the whole training set. Therefore, a fully developed tree may suffer from over-fitting. Both Gradient Tree Boosting and Random Forest try to account for this potential drawback. They use smaller subsets of the original data to train individual trees which helps with preventing overfitting. Random Forest does this by bagging (bootstrap aggregating), while Gradient Tree Boosting achieves this by boosting.

More specifically, Random Forest [7] works as a large collection of decision trees, each calculated by a random subset of inputs. The most simple reduction method to obtain a single value result is a modest incident ranking returning the most frequent solution or averaging the output. On the other hand, Gradient Tree Boosting [17] trains the different trees in a sequential order. Therefore, it can analyze the performance and choose the training subset according to minimize the error.

Additionally, we use Gaussian Mixture Models as fourth prediction formalism. They are stochastic models based on a mix of probability distributions. They capture a target distribution using superposition of multiple Gaussian distributions, adjusting their means and covariances. Mixture models are used as an approach for situations when a single Gaussian distribution is unable to capture complex data, whereas a linear superposition of two or more Gaussians gives a better characterization of the data set. By using a sufficient number of Gaussian distributions and by adjusting their means and covariances as well as the coefficients in the linear combination, almost any continuous density can be approximated to arbitrary accuracy.

The following function describes this effect:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

$\mathcal{N}(x|\mu_k, \Sigma_k)$ = Gaussian component
$\mu_k$ = component mean
$\Sigma_k$ = covariance of a component
$\pi_k$ = mixing coefficient
$k$ = number of components

Gaussian mixture models are a candidate for power prediction in this work, as they have been used to model power consumption of server environments in the past [14]. By using Gaussian Mixture Vector Quantization (GMVQ) based training and classification with vector quantization using the "nearest-neighbor" approach, the previous trained data serves as classification information for power prediction.

We use the implementation of the Smile [28] library for Java for all of the considered formalisms.

## 4.3 Interpolating Measurement Results

We require the same amount of measurements for each worklet used to construct the regressor matrix $X$. However, SERT's default settings measure different worklets with different load level counts, as described in Section 3.1. We consider two options to tackle this issue: zero padding and interpolation. Both methods are intended to create artificial results at the missing load levels for the respective worklets.

*Zero padding* simply fills the gaps in load levels for worklets with too few levels with results containing the value 0. This method is primarily useful if these results are needed for mathematical correctness, but otherwise discarded later in the prediction process (i.e., when the prediction method in question does not use them). Otherwise, the expectation is that zero padding enables the prediction to run, but with negative effects on prediction accuracy.

*Interpolation* creates missing results based on the neighboring load levels' results. In general, it is the reconstruction of a continuous function $f(x)$ from $n$ different sample points: $\{(x_1, f_1), (x_2, f_2), ..., (x_n, f_n)\}$.

We investigate the use of three interpolation methods:

- *Nearest Neighbor Interpolation* interpolates results by returning the closest existing result: $f(x) = f(x_i)$ with $x_i \in \{x_1, ..., x_n\}$ being the nearest neighbor to $x$, meaning that $\forall x_j \in \{x_1, ..., x_n\} : |x - x_i| \leq |x - x_j|$.
- *Linear Interpolation* interpolates using a linear function $f(x) = f(x_i) + (f(x_{i+1}) - f(i))\frac{x - x_i}{x_{i+1} - x_i}$ given the two nearest neighbors of $x$, $x_i$ and $x_{i+1}$, with $x_i \leq x$ and $x_{i+1} > x$.

- The *cross-validation interpolation* approach of [42], which selects an interpolation method from a range of polynomial and scattered interpolation methods (including nearest neighbor, linear, and higher order polynomials) using cross validation. Note that this approach requires a minimum of three points of data in a set to be interpolated. This is an issue in our use-case, as some SERT worklets only feature two datapoints. We have to discard these worklets when using this interpolation, whereas nearest neighbor and linear interpolation are capable of working on the smaller sets.

## 4.4 Self-Prediction Accuracy

Our power prediction model uses parameter optimization and cross-validation of the different underlying modeling formalisms in order to increase the prediction's accuracy. During this optimization phase, each potential optimization candidate is evaluated and either discarded or adopted. We use the model's self-prediction error for this run-time evaluation. We then predict the training target application power consumption values and compare the resulting errors. We allow the use of several error metrics for this comparison and consider two, specifically:

(1) *Root Mean Squared Error*:

$$e_{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i' - y_i)^2}{n}}.$$

(2) *Mean Absolute Percentage Error*:

$$e_{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{\|y_i' - y_i\|}{y_i}.$$

## 4.5 Parameter Modeling and Optimization

The prediction formalisms used by our offline prediction approach feature several different formalism-specific parameter settings that can affect the prediction's accuracy. Specifically, we consider the following parameters:

We optimize these parameters automatically using a method inspired by [30]. We create a generic parameter model, which sets the optimization exploration space for each paramter by assigning it a *minimum* and *maximum* value and an initial exploration *step* size.

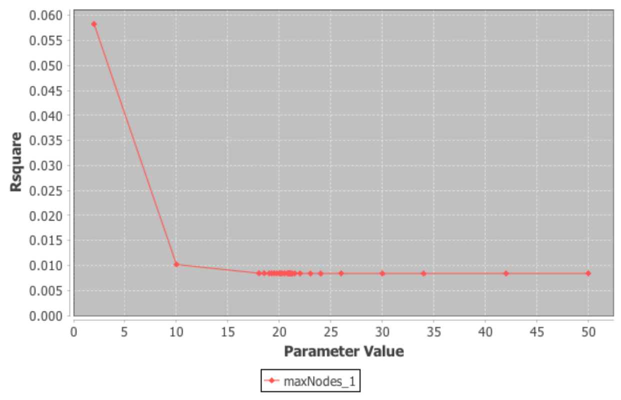

**Figure 2: Example of Local Search Parameter Optimization**

We then apply an iterative local search for each parameter. We split the parameter's search space between its minimum and maximum values into $k+1$ eqal parts, resulting from the parameter's step size. We then calculate the self-prediction error of the model with the parameter under consideration for each of the potential values. We then create local search spaces around each local minimum, and iteratively explore those search spaces with a halved step size. This process repeats until a maximum search depth $d$ is achieved, at which point the smallest local minimum is picked. Figure 2 shows an example of the local search for the `max nodes` parameter in a regression tree. The figure shows that the local search analyzes potential parameter values around 20 in greater detail, trying to find the minimum R sqare error. We perform the parameter value search for each potential parameter. This is then repeated iteratively for all parameters for $n$ iterations, where $n$ the number of parameters per default.

## 4.6 Prediction Formalism Selection

Finally, we explore if it is possible to select the concrete prediction formalism automatically. The power prediction approach uses all prediction formalisms introduced in Section 4.2. The idea for the automated selection is to not fix a modeling formalism, but instead select it automatically at run-time. To to this, we first optimize the parameters for all potential formalisms using our iterative optimization method from Section 4.5. We then calculate the self-prediction RMSE for all of the optimized formalisms and select the one with the lowest error for prediction. The entire approach from data processing to parameter optimization and formalism selection approach is illustrated in Figure 1.

## 5 EVALUATION

We evaluate our offline prediction model using three target applications and three different real-world, physical servers. We run the SPEC SERT on each of those servers to measure and characterize its power consumption. We then predict the target applications' power consumption when running on two of those servers using the respective target server's SERT and a training set consisting of a SERT result and target application power measurements from one of the servers. We compare measured application power consumption with the predicted consumption for all measured load levels and servers and calculate the aggregate prediction error using the mean absolute percentage error (MAPE).

$$e_{MAPE} = \frac{\sum_{l \in loadlevels} \frac{|pwr_{predicted}(l) - pwr_{measured}(l)|}{pwr_{measured}(l)}}{|loadlevels|}. \quad (4)$$

In addition to the server-specific MAPE of Eq. 4, we also consider the aggregate MAPE over all servers for a specific configuration of our offline power prediction for a certain application.

We consider the following three applications:

- **Pi**: A worklet that ships with the SPEC ChauffeurWDK [1] and computes Pi by calculating up to 100000 iterations of the Gregory-Leibniz series.
- **Friendgraph**: Friendgraph is supposed to emulate a simple social network graph of "friends", which store arbitrary numeric properties within a matrix. A transaction calculates

**Table 1: System under characteristics measurec**

| | 4 |
|---|---|
| **Model** | HP ProI I |
| **CPU** | 4 c |
| Xeon Model | E3-123 |
| Clock | 3.4 |
| Generation | Sky |
| **Memory** | 2 x |
| **Storage** | 1 x 46 |
| **Idle** Pwr | 28 |
| **Max** Pwr | 106 |

"friend"-value by agg first and second-ord

- **Dell DVD Store** [1 application develope tion with a MariaDB generate load using pattern. Users log ir cart, and then log out.

All servers in our experiment run Debian 9.4 (Kernel 4.9.82), with Docker 18.03.0-ce, and Java HotSpot 64-Bit (build 25.161-b12). Table 1 shows the hardware configuration of our three testing devices, which we identify using the core-count.

## 5.1 Measuring Target Application Power and Performance

We measure the target application power consumption and performance using the SPEC Power and Performance Benchmark Methodology [36]. We do this by implementing our workloads within the SPEC ChauffeurWDK [1] or by implementing a load driver in Chauffeur to drive the external workload in case of the DVD Store. Using this methodology and implementation, we measure the target applications' power and performance (throughput) at four target load levels: 25%, 50%, 75%, and 100%. We connect an external power measurement device to the AC power inlet of the SUT. An external director machine controls the experiment using a network connection to the SUT. The SUT runs Chauffeur's *host*-software, which launches client processes that execute the workload or delegate it to the DVD Store using HTTP. Figure 3 shows the device setup.

Analogue to how SERT operates, we perform a warmup run for 30 seconds and then calibrate the maximum load level of the target application by running it on parallel on every available hardware thread. We repeat this calibration two times, with each separate run having a pre-measurement duration of 15 seconds, a measurement duration of 120 seconds, and a post-measurement duration of 15 seconds.

After calibration, we measure performance and power consumption for the target load levels in descending order. The load levels' pre-measurement, post-measurement, and measurement times are equal to the calibration. Our reported per-load-level power and performance results are the average of the 120 second measurements.

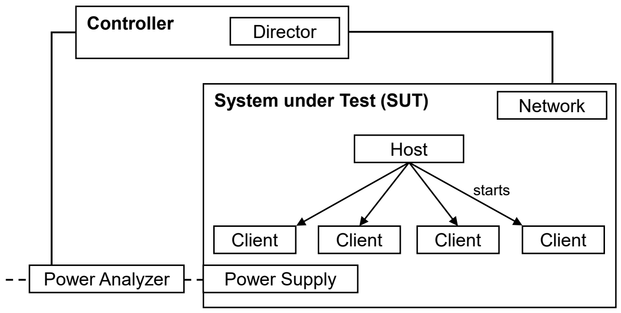

**Figure 3: Device setup for power target application power and performance measurements**

Figure 4 shows the calibration and measurement intervals with their respective duration. The maximum coefficient of variation during our measurement runs is the performance variation of the Dell DVD Store at full load on the 10 core machine. It features a CV of 1.6%, which is well below the maximum boundary of 5%, defined by SPEC.

## 5.2 Unoptimized Power Prediction

First, we analyze the accuracy of our prediction formalisms without interpolation, optimization, and formalism selection. We compare our following methods against this unoptimized variant to classify their respective improvement. When not using interpolation, we only use worklets with at least four load levels, meaning that the memory and storage worklets are discarded. For the SSJ worklet, we discard the four of the eight load levels that do not appear in the other worklets.

Considering that parameter optimization is removed, we set each of the regression formalism's parameters to the minimum parameter within our parameter space. Specifically, Regression Tree's `max nodes` and `node size` are set to 2 and RandomForest and GradientTreeBost both start with a single tree. Finally, GaussianRegression has its `shrinkage` set between 0 and 1 and its `kernel width` between 0.1 and 40. Table 2 shows the prediction errors of this unoptimized method.

The results in Table 2 show differences in the prediction accuracies of the underlying formalisms. Yet, even the more accurate methods suffer from poor accuracy when not optimized and without any interpolation. In general, the regression mechanisms seem to be able to capture the problem with some error. Gradient Tree Boost and Random Forest both feature a mean absolute percentage error of slightly more than 30%. Even though these two formalisms achieve similar average results, they differ in terms of variance. Gradiant Tree Boost's MAPE has a standard deviation of 11.8%,

**Table 2: Unoptimized prediction errors of base formalisms.**

| **Formalism** | **Avg. MAPE** |
|---|---|
| Gradient Tree Boost | 32.94% |
| Regression Tree | 19.42% |
| Random Forest | 31.03% |
| Gaussian Regression | 99.99% |

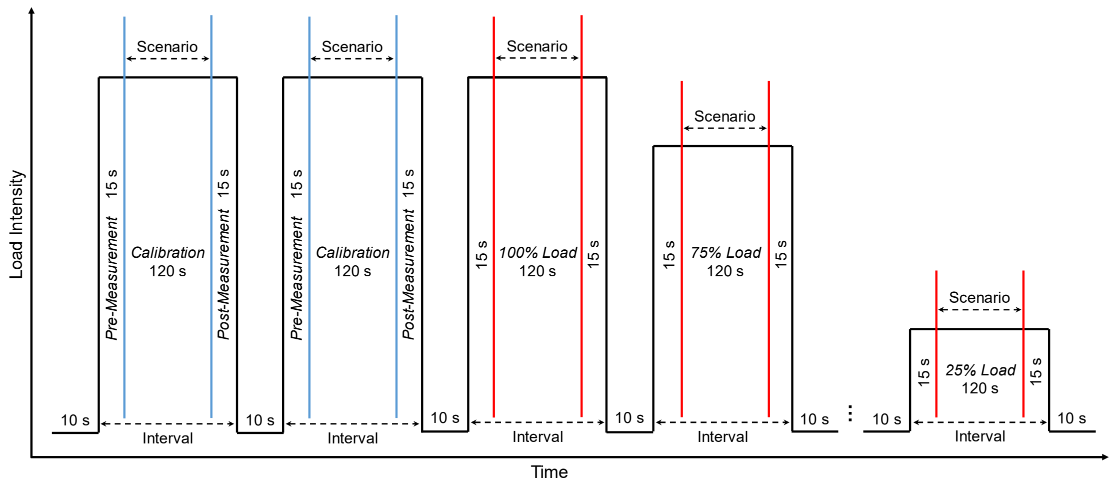

**Figure 4: Measurement intervals for target application power and performance measurements**

whereas the Random Forest has a lower variation of 7.7%. Both methods can not achieve the accuracy of the regular Regression Tree, though. It has an average MAPE of 19.42% with a standard deviation of only 2.6%.

We attribute this to the relatively little amount of training data available. Four load levels per worklet do not provide sufficient data for an accurate training of all formalisms. The Gaussian Regression is most affected by this. Seven worklets with at least four load levels plus idle seems to not provide sufficient data to derive multiple Gaussian distributions. Consequently, we do not consider Gaussian Mixture Models in the following tests.

## 5.3 Predicting Power using Interpolation and Optimization

Our unoptimized tests indicate that the amount of data provided by SERT results is too little for most prediction methods. Consequently, we enable interpolation in conjunction with our prediction formalism parameter optimization method. The interpolation mechanism ensures that no measurement must be discarded in worklets with high load level counts. Specifically, it creates artificial load levels for all worklets with fewer levels than SSJ, which is the worklet featuring the most load levels. As a result, all worklets are padded to a total of nine load levels (eight SSJ load levels plus idle power). We use all worklets with a minimum of four load levels, enabling the use of adaptive interpolation. Parameter optimization is left at default settings, meaning that it uses as many iterations as the underlying method has parameters for optimization. Our regression formalisms feature two parameters, each, resulting in an iteration count of two.

**Table 3: Formalism prediction error with adaptive interpolation and parameter optimization.**

| Formalism | Avg. MAPE |
|---|---|
| Gradient Tree Boost | 32.94% |
| Regression Tree | 10.89% |
| Random Forest | 32.05% |

The prediction accuracy of our three regression formalisms used with adaptive load-level interpolation and parameter optimization is shown in Table 3. Gradient Tree Boost' accuracy improves, but only by 0.004% percentage points, which is not significant. Random Forest's accuracy even decreases by 1.2% points. However, Regression Tree improves significantly. With optimization and interpolation enabled, it features an prediction error of 10.89%, which is an improvement of 8.5 percentage points and relative improvement of 43.9% compared to our unoptimized experiment. Figure 5 shows the measured and predicted power consumption of the Pi worklet on the two target servers. It shows that the prediction is generally accurate, except for the 100% load level on the 10 core

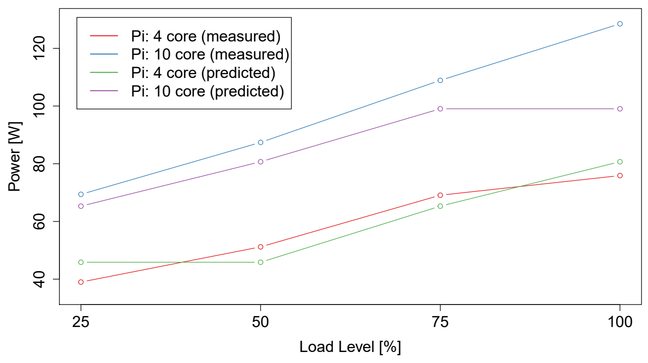

**Figure 5: Measurements and Regression Tree prediction of Pi worklet power**

**Table 4: Formalism prediction error with parameter optimization, but not using any interpolation.**

| Formalism | Avg. MAPE |
|---|---|
| Gradient Tree Boost | 32.94% |
| Regression Tree | 19.42% |
| Random Forest | 32.26% |

server. The prediction expects a decline in power consumption increase, which does not occur on the 10 core device.

The amount of data provided by interpolated results seems to be sufficient for a tuned regression tree. In contrast, the other two regression methods do not improve significantly, which leads to conclude that the interpolated SERT results did still not produce sufficient data for training of accurate models for these formalisms.

## 5.4 Parameter Optimization and no Interpolation

The predictions using parameter optimization and interpolation in Section 5.3, significantly improve the prediction accuracy of the unoptimized formalisms. The part played by the parameter optimization and interpolation, respectively, in this improvement remains an open question. To investigate if parameter optimization caused this improvement by itself, we test our prediction formalisms using parameter optimization only. That means that we use four load levels only, discarding excess SSJ load levels and worklets with fewer load levels, similar to to unoptimized approach in Section 5.3.

Table 4 shows the prediction errors of using parameter optimization without interpolation. In general, the differences to unoptimized prediction are very small. Gradient Tree Boost and Regression Tree do, in fact, see no change. Random Forest's accuracy, on the other hand, decreases. This would indicate that parameter optimization is of no help for such a small dataset and the chosen prediction formalisms. However, accuracy increased in the results of Section 5.3, where optimization was used. We investigate this further in the following tests.

## 5.5 Interpolation with Unoptimized Parameters

Our tests using optimization without any interpolation in Section 5.4 do not show any significant improvement in prediction accuracy. Consequently, we investigate if interpolation achieves the improvement of Section 5.3 by itself. To test this, we again apply adaptive interpolation to all worklets with four load levels in order to match the load level count of SSJ (eight plus idle). We also, again, interpolate the reference workload to nine load levels.

**Table 5: Formalism prediction error with interpolation, but not using any parameter optimization.**

| Formalism | Avg. MAPE |
|---|---|
| Gradient Tree Boost | 32.91% |
| Regression Tree | 20.40% |
| Random Forest | 30.41% |

Table 5 shows the prediction accuracy when using the unoptimized parameter set on interpolated training and prediction data. Interestingly, these results do also not differ significantly from the unoptimized prediction. Gradient Tree Boost and Random forest improve marginally by less than 1 percentage point. Due to the lack of optimization, Regression Tree's accuracy decreases when using interpolation without any parameter optimization.

These results indicate that neither parameter optimization nor interpolation increase prediction accuracy by themselves. In addition, they disprove the potential conclusion from the optimization-only results in Section 5.4 that parameter optimization does not improve prediction accuracy and improvements are instead due to interpolation. Instead, the separate tests using interpolation and parameter optimization only clearly indicate that the combination of both is responsible for the improvements in prediction accuracy. Interpolation ensures that sufficient data is available to run optimization against and parameter optimization configures the prediction formalisms for the interpolated dataset.

## 5.6 Prediction Accuracy depending on Interpolation Method

Our results in Section 5.3 show that interpolation helps in increasing prediction accuracy, particularly when using the Regression Tree prediction formalism. Those experiments used an adaptive interpolation approach to generate additional data for training. We investigate the choice of the specific interpolation method and its impact on accuracy. We focus on the Regression Tree prediction formalisms, as it shows the greatest sensitivity to interpolation and parameter optimization in addition to usually providing the most accurate predictions. Table 6 shows the prediction errors of using Regression Tree with parameter optimization based on datasets prepared with the different interpolation methods.

We investigate *zero padding* as an alternate approach to unoptimized prediction and to interpolation. Zero padding of missing results allows us to not discard excess SSJ load levels and still meet the constraints of the prediction methods. As seen in Table 6, using zero padding results in a Regression Tree prediction error of 19.91%, which is slightly less accurate than simple, unoptimized prediction. In contrast, the prediction errors when using the interpolation methods are all significant improvements compared to unoptimized prediction. *Nearest Neighbor Interpolation* features the smallest improvement in accuracy, improving the average prediction MAPE by only 3.92 percentage points. Adaptive Interpolation improves the prediction by 8.5 percentage points, which is a relative improvement of 43.9% compared to our unoptimized experiment.

**Table 6: Regression Tree prediction error depending on interpolation method.**

| Interpolation Method | Avg. MAPE |
|---|---|
| Zero Padding | 19.91% |
| Nearest Neighbor | 15.49% |
| Linear Interpolation | 9.49% |
| Adaptive Interpolation | 10.89% |

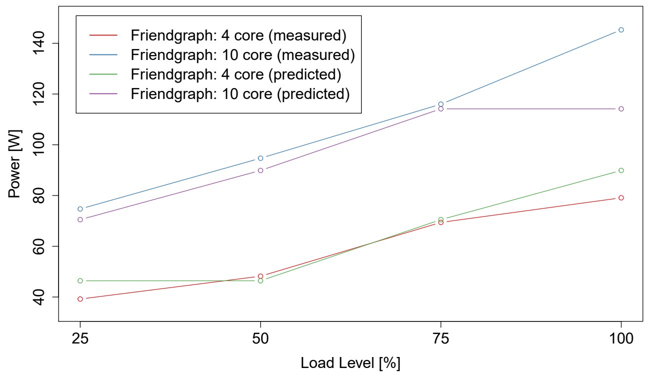

**Figure 6: Measurements and Regression Tree prediction (using linear interpolation) of Friendgraph worklet power**

Interestingly, linear interpolation provides the most accurate predictions with a MAPE of 9.49%. This is a relative improvement of 51% compared to the unoptimized Regression Tree prediction error. Again, we surmise that the small size of the dataset is a cause for this, as the adaptive interpolation has too few datapoints (usually four) to adapt itself. Instead, simply picking an interpolation option that always delivers good results and is applicable to such small sets, seems to lead to more accurate predictions. Figure 6 shows the results of power prediction for the Friendgraph worklet using linear interpolation. As a sidenote, including the storage worklets in predictions using linear interpolation does not increase prediction accuracy, leading to a prediction error of 14.09%.

## 5.7 Including the Storage Worklets

The adaptive interpolation requires a minimum number of four data-points per function to interpolate. Consequently, it is not possible to apply this interpolation method for the two storage worklets, which feature only two load levels, each. However, when using linear interpolation or nearest neighbor interpolation, we gain the ability to include these two worklets for training and prediction. Table 7 shows the prediction errors. Including the storage worklet does not improve prediction accuracy and even decreases it for the dataset with linear interpolation.

We attribute this to two factors: The number of data-points and the power scaling of the storage worklets. The storage worklets feature an almost constant power consumption over the load levels with only a little increase that is poorly modeled due to only having two points of measurement per worklet. This has as a consequence that they offer no valuable information for the power prediction formalisms and thus their use does not offer an increase in prediction accuracy.

**Table 7: Regression Tree prediction error with storage worklets included.**

| Interpolation Method | Avg. MAPE |
|---|---|
| Nearest Neighbor | 15.49% |
| Linear Interpolation | 14.09% |

**Table 8: Formalism prediction error compared to interal RMSE for formalism selection.**

| Formalism | Self-Prediction RMSE | Avg. MAPE |
|---|---|---|
| Gradient Tree Boost | 24.405 | 32.94% |
| Regression Tree | 5.849 | 10.89% |
| Random Forest | 24.399 | 32.05% |

## 5.8 Prediction Formalism Selection

Finally, we investigate if our prediction formalism selection method is able to choose the correct formalism. To this end, we compare the internal self-prediction RMSE that the selection method uses to the MAPE of the actual predictions to assess the selection's accuracy. Again, we use our regression formalisms with adaptive interpolation and parameter optimization.

Table 8 shows that the internal self-prediction RMSE results in the same ranking as our external prediction MAPE. Regression Tree is considered the by far best method using both metrics, whereas Random Forest beats Gradient Tree Boost barely. However, the entire evaluation also indicates that formalism selection is not necessary, as Regression Tree seems to always offer the best prediction accuracy.

## 5.9 Evaluation Conclusion

Our evaluation shows that our offline power prediction method can accurately predict the power consumption of a target server using SERT results. It also shows that the combination of parameter optimization and interpolation can significantly increase the prediction accuracy compared to unoptimized formalisms by as much as 51%, even though both interpolation and parameter optimization do not achieve significant improvements in accuracy on their own.

## 6 CONCLUSIONS

This paper presents an offline power prediction approach that is able to predict the power consumption using data provided by the SERT standard rating tool, which is employed by the U.S. EPA Energy Star Program. The approach predicts the power consumption of an application for multiple load levels on a target server, for which only the SERT results need be provided with an average error of 9.49%, which is an improvement of 51% compared to unoptmized methods.

The results in this paper help server operators choose the least power consuming server for their applications. It can be combined with performance models for a prediction of energy efficiency or can be used as part of a server power and performance constraint optimization.

### 6.1 Future Work

In this paper, we consider Gaussian Regression, Gradient Tree Boost, Regression Tree, and Random Forest as concrete regression mechanisms to use in our prediction. Among these methods, Random Forest seems to produce the best results. In the future, our prediction method would be enhanced by adding and evaluating additional regression mechanisms. These mechanisms might help to increase prediction accuracy for some or all datasets.

In addition, our method could also be enhanced if it were able to use other publically available results for the servers under consideration. E.g., it might be extended to also use results provided by SPECpower_ssj2008 [23]. This data could be combined wit hthe data of the SPEC SERT to provide larger dataset, increasing prediction accuracy and enabling the use of other regression mechanisms that benefit from larger datasets. Use of other benchmark results could also enable prediction of other workload types. E.g., the inclusion of results from GPGPU benchmarks might enable prediction of power consumption for GPGPU applications.

### 6.2 Threats to Validity

We hypothesize in Section 5.8 that formalism selection works well, as the self-prediction RMSE results in the same ranking as the Avg. MAPE of the actual predictions. Additionally, we observe that Regression Tree is the best method in all evaluation scenarios and that a formalism selection was therefore not required on our test set. However, we do not claim that these results are transferable to other case studies as our evaluation test size was not big enough to draw such conclusion. Instead, our goal was to show the feasibility of the proposed approach and to evaluate its effectiveness, rather than to draw any further conclusions. Future research will hopefully show if an automatic selection technique is actually required in this domain.

Additionally, we show that parameter optimization (together with interpolation) improves the prediction accuracy compared to using the default parameters (see Section 5.3). However, one remaining question is if the found parameters are similar across different (sub-) data sets. One could investigate if there is a set of optimal parameters in our domain, or if the parameter optimization process has to be redone on every data set. One research question could be to investigate the difference between the default parameters, our optimized set and the optimum on a different evaluation set.

## ACKNOWLEDGMENT

This work was funded by the German Research Foundation (DFG) under grant No. (KO 3445/11-1) and was supported by the SPEC Research Power Working Group.

## REFERENCES

[1] Jeremy Arnold. 2013. *Chauffeur: A framework for measuring Energy Efficiency of Servers.* Master Thesis. University of Minnesota.

[2] C. Babcock. 2012. NY Times data center indictment misses the big picture. *InformationWeek Cloud.*

[3] L.A. Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec 2007), 33–37. https://doi.org/10.1109/MC.2007.443

[4] Robert Basmadjian, Nasir Ali, Florian Niedermeier, Hermann de Meer, and Giovanni Giuliani. 2011. A Methodology to Predict the Power Consumption of Servers in Data Centres. In *Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking (e-Energy '11).* ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2318716.2318718

[5] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. 2012. Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. *Future Gener. Comput. Syst.* 28, 5 (May 2012), 755–768. https://doi.org/10.1016/j.future.2011.04.017

[6] W. L. Bircher and L. K. John. 2012. Complete System Power Estimation Using Processor Performance Events. *IEEE Trans. Comput.* 61, 4 (April 2012), 563–577. https://doi.org/10.1109/TC.2011.47

[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA.

[9] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. *SIGARCH Comput. Archit. News* 28, 2 (May 2000), 83–94. https://doi.org/10.1145/342001.339657

[10] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18).* ACM, New York, NY, USA, 41–42. https://doi.org/10.1145/3185768.3185771

[11] Xi Chen, Chi Xu, Robert P. Dick, and Zhuoqing Morley Mao. 2010. Performance and Power Modeling in a Multi-programmed Multi-core Environment. In *Proceedings of the 47th Design Automation Conference (DAC '10).* ACM, New York, NY, USA, 813–818. https://doi.org/10.1145/1837274.1837479

[12] G. Contreras and M. Martonosi. 2005. Power prediction for Intel XScale/spl reg/ processors using performance monitoring unit events. In *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.* 221–226. https://doi.org/10.1109/LPE.2005.195518

[13] Dell, Inc. 2011. The DVD Store Version 2. (December 2011). http://en.community.dell.com/techcenter/extras/w/wiki/dvd-store, last accessed May 2018.

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

[15] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. 2006. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS).*

[16] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz AndrÃľ Barroso. 2007. Power Provisioning for a Warehouse-sized Computer. In *The 34th ACM International Symposium on Computer Architecture.* http://research.google.com/archive/power_provisioning.pdf

[17] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[18] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir. 2002. Using complete machine simulation for software power estimation: the SoftWatt approach. In *Proceedings Eighth International Symposium on High Performance Computer Architecture.* 141–150. https://doi.org/10.1109/HPCA.2002.995705

[19] Intel® Corporation 2018. *Intel® 64 and IA-32 Architectures Software Developer's Manual.* Intel® Corporation.

[20] Canturk Isci and Margaret Martonosi. 2003. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36).* IEEE Computer Society, Washington, DC, USA, 93–. http://dl.acm.org/citation.cfm?id=956417.956567

[21] Gueyoung Jung, M.A. Hiltunen, K.R. Joshi, R.D. Schlichting, and C. Pu. 2010. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on.* 62–73. https://doi.org/10.1109/ICDCS.2010.88

[22] A. B. Kahng, Bin Li, L. S. Peh, and K. Samadi. 2009. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *2009 Design, Automation Test in Europe Conference Exhibition.* 423–428. https://doi.org/10.1109/DATE.2009.5090700

[23] K.-D. Lange. 2009. Identifying Shades of Green: The SPECpower Benchmarks. *Computer* 42, 3 (March 2009), 95–97. https://doi.org/10.1109/MC.2009.84

[24] K.-D. Lange and Michael G. Tricker. 2011. The Design and Development of the Server Efficiency Rating Tool (SERT). In *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (ICPE '11).* ACM, New York, NY, USA, 145–150. https://doi.org/10.1145/1958746.1958769

[25] Klaus-Dieter Lange, Mike G. Tricker, Jeremy A. Arnold, Hansfried Block, and Christian Koopmann. 2012. The Implementation of the Server Efficiency Rating Tool. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12).* ACM, New York, NY, USA, 133–144. https://doi.org/10.1145/2188286.2188307

[26] Benjamin C. Lee and David M. Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. *SIGPLAN Not.* 41, 11 (Oct. 2006), 185–194. https://doi.org/10.1145/1168918.1168881

[27] Adam Lewis, Soumik Ghosh, and N.-F. Tzeng. 2008. Run-time Energy Consumption Estimation Based on Workload in Server Systems. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems (HotPower'08).* USENIX

Association, Berkeley, CA, USA, 4–4. http://dl.acm.org/citation.cfm?id=1855610.1855614

[28] Haifeng Li. [n. d.]. Smile - Statistical Machine Intelligence and Learning Engine. https://haifengl.github.io/smile/index.html. ([n. d.]). Last accessed: September 2019.

[29] Min Yeol Lim, Allan Porterfield, and Robert Fowler. 2010. SoftPower: Fine-grain Power Estimations Using Performance Counters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*. ACM, New York, NY, USA, 308–311. https://doi.org/10.1145/1851476.1851517

[30] Qais Noorshams, Dominik Bruhn, Samuel Kounev, and Ralf Reussner. 2013. Predictive Performance Modeling of Virtualized Storage Systems using Optimized Statistical Regression Techniques. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE 2013) (ICPE'13)*. ACM, New York, NY, USA, 283–294. https://doi.org/10.1145/2479871.2479910

[31] Meikel Poess, Raghunath Othayoth Nambiar, Kushagra Vaid, John M Stephens Jr, Karl Huppler, and Evan Haines. 2010. Energy benchmarks: a detailed analysis. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. ACM, 131–140.

[32] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. 2008. A Comparison of High-level Full-system Power Models. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems (HotPower'08)*. USENIX Association, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=1855610.1855613

[33] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. 2007. JouleSort: A Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*. ACM, New York, NY, USA, 365–376. https://doi.org/10.1145/1247480.1247522

[34] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu. 2013. A Study on the Use of Performance Counters to Estimate Power in Microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60, 12 (Dec 2013), 882–886. https://doi.org/10.1109/TCSII.2013.2285966

[35] Karan Singh, Major Bhadauria, and Sally A. McKee. 2009. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News* 37, 2 (July 2009), 46–55. https://doi.org/10.1145/1577129.1577137

[36] Standard Performance Evaluation Corporation. [n. d.]. SPEC Power and Performance Benchmark Methodology. http://spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf. ([n. d.]).

[37] Christian Stier, Anne Koziolek, Henning Groenda, and Ralf Reussner. 2015. Model-Based Energy Efficiency Analysis of Software Architectures. In *Proceedings of the 9th European Conference on Software Architecture (ECSA '15) (Lecture Notes in Computer Science)*. Springer. https://doi.org/10.1007/978-3-319-23727-5_18

[38] Yuan Tian, Chuang Lin, and Min Yao. 2012. Modeling and analyzing power management policies in server farms using Stochastic Petri Nets. In *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*. 1–9.

[39] Ghislain Landry Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. 2014. Exploiting performance counters to predict and improve energy performance of HPC systems. *Future Generation Computer Systems* vol. 36 (July 2014), pp. 287–298. https://doi.org/10.1016/j.future.2013.07.010

[40] R. Urgaonkar, U.C. Kozat, K. Igarashi, and M.J. Neely. 2010. Dynamic resource allocation and power management in virtualized data centers. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*. 479–486. https://doi.org/10.1109/NOMS.2010.5488484

[41] Jóakim von Kistowski, Maximilian Deffner, and Samuel Kounev. 2018. Run-time Prediction of Power Consumption for Component Deployments. In *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC 2018)*.

[42] Jóakim von Kistowski and Samuel Kounev. 2015. Univariate Interpolation-based Modeling of Power and Performance. In *Proceedings of the 9th EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2015)*. https://doi.org/10.4108/eai.14-12-2015.2262579

[43] Joakim von Kistowski, Marco Schreck, and Samuel Kounev. 2016. Predicting Power Consumption in Virtualized Environments. In *Computer Performance Engineering: 13th European Workshop, EPEW 2016, Chios, Greece, October 5-7, 2016, Proceedings*, Dieter Fiems, Marco Paolieri, and N. Agapios Platis (Eds.). Springer International Publishing, Cham, 79–93. https://doi.org/10.1007/978-3-319-46433-6_6

[44] T.A Welch. 1984. A Technique for High-Performance Data Compression. *Computer* 17, 6 (June 1984), 8–19. https://doi.org/10.1109/MC.1984.1659158