# Towards a Predictive Energy Model for HPC Runtime Systems Using Supervised Learning

**Conference Paper** · August 2019

DOI: 10.1007/978-3-030-48340-1_48

CITATIONS
5

READS
326

**7 authors**, including:

Neda Davoudi
Technische Universität München
**6** PUBLICATIONS **119** CITATIONS
SEE PROFILE

Gabrielle Poerwawinata
University of Basel
**4** PUBLICATIONS **8** CITATIONS
SEE PROFILE

Matthias Maiterth
Ludwig-Maximilians-University of Munich
**7** PUBLICATIONS **217** CITATIONS
SEE PROFILE

Alessio Netti
Intel
**27** PUBLICATIONS **74** CITATIONS
SEE PROFILE

# Towards a Predictive Energy Model for HPC Runtime Systems Using Supervised Learning

Gence Ozer[1], Sarthak Garg[1], Neda Davoudi[1], Gabrielle Poerwawinata[1], Matthias Maiterth[2], Alessio Netti[3,1], and Daniele Tafani[3]

[1] Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
{gence.ozer,sarthak.garg,neda.davoudi,gw.poerwawinata}@tum.de
[2] Intel Deutschland GmbH, Dornacher Str. 1, 85622 Feldkirchen, Germany
matthias.maiterth@intel.com
[3] Leibniz-Rechenzentrum, Boltzmannstr. 1, 85748 Garching, Germany
{alessio.netti,daniele.tafani}@lrz.de

**Abstract.** High-Performance Computing systems collect vast amounts of operational data with the employment of monitoring frameworks, often augmented with additional information from schedulers and runtime systems. This amount of data can be used and turned into a benefit for operational requirements, rather than being a data pool for post-mortem analysis. This work focuses on deriving a model with supervised learning which enables optimal selection of CPU frequency during the execution of a job, with the objective of minimizing the energy consumption of a HPC system. Our model is trained utilizing sensor data and performance metrics collected with two distinct open-source frameworks for monitoring and runtime optimization. Our results show good prediction of CPU power draw and number of instructions retired on realistic dynamic runtime settings within a relatively low error margin.

**Keywords:** Energy Efficiency · Monitoring Systems · Random Forest · DVFS · Runtime Systems.

## 1 Introduction

The primary goal of High-Performance Computing (HPC) centers is to provide computational resources to their users, a feat that is paid proportional to the center's size in terms of energy consumption. Over the past decade, relevant concerns have arisen for the massive amount of power necessary for operating such systems at all levels, from the building infrastructure, past the hardware and software layers, to the application code run by users [6,7,5,18]. Significant emphasis has been placed into optimization of the software stack layer, particularly with the development and adoption of comprehensive sensor monitoring tools and efficient optimization mechanisms for scheduling and runtime systems [4,2,1]. Improvements in processor design made CPU-level measurements and tuning widely available, thanks to different power management techniques, such as Dynamic Voltage and Frequency Scaling (DVFS) and the Intel's Running Average

Power Limit (RAPL) interface. Combined with fine-granularity acquisition of sensor and hardware counter data, the features offered by these technologies can support both scheduling systems and runtime frameworks. The measurements are mapped to characteristics of the running application to adapt operational modes (i.e., selecting an optimal CPU frequency) appropriate for optimizing objectives such as power consumption, thus reducing operational costs. In this regard, machine learning techniques have been very promising for forecasting and evaluating hardware metrics and ultimately optimize such decisions [16,19].

*Related Work.* Amongst the different proposed machine learning techniques to forecast hardware metrics for power management, reinforcement learning is often regarded as one of the most promising and widely adopted [10,12,15]. Despite its advantages, as the state-space expands exponentially with the core count, most of the traditional models trained with this technique are limited to a single processor and a small number of cores. Improvements in this direction have been achieved using modular reinforcement learning [16]. However, the training process in these models is *online*, which is significantly challenging outside of a full system simulator. In this regard, alternative approaches based on supervised learning introduce less overhead and may be more suitable for implementations in realistic scenarios. Yang et. al.[19] developed a runtime model using linear regression to map an application task on a computing resource during runtime, ensuring minimum energy consumption for a given application performance requirement. A branch of research employs time series analysis to characterize the history of an application and forecast its behavior. This research shows that time series analysis with Autoregressive Moving Average and Singular Spectrum Analysis can be used with runtime traces to achieve good prediction [8]. Kunkel et al. [9] analyzed the quality of monitoring data and applied Principal Component Analysis to identify the counters required for power prediction.

Wang et. al. [14] have built and evaluated a model for performance prediction for power-capped applications. This approach is very valuable and shows the feasibility of modeling approaches of performance with variable power. Tuncer et al. [13] applied supervised machine learning for anomaly detection using statistical features of monitoring data. While our model is similar, its scope is different: the former work focuses on classification of system states at coarse time scales, while we perform time series regression at a very fine scale, in the order of milliseconds, using high-fidelity data both from monitoring and runtime systems.

*Contributions.* In this paper we propose a case study for predicting hardware metrics with supervised learning in HPC systems. Specifically, we derive a machine learning model based on random forest regression focusing on the prediction of *CPU power* and total number of *instructions retired*. Data acquisition is performed by using two distinct data sources, specifically the Intel GEOPM runtime framework and the Data Center Database (DCDB) monitoring tool of the Leibniz Supercomputing Center (LRZ). We demonstrate that our model is capable of predicting the selected hardware metrics with high accuracy. Results are obtained by executing a set of benchmarks covering a wide range of HPC

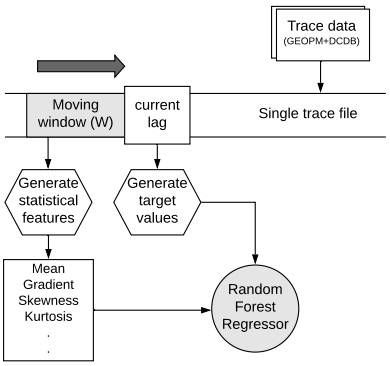



**Fig. 1.** Training (offline)

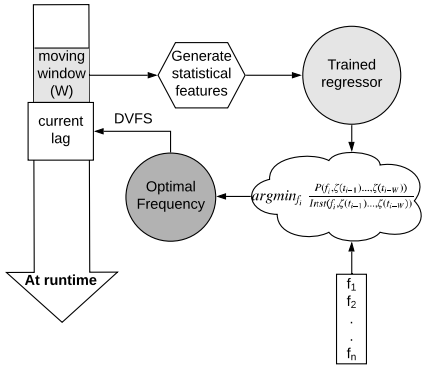



**Fig. 2.** Inference (at runtime)

application behaviours of interest to LRZ in terms of compute intensity and memory utilization. We then propose a theoretical model to use the predictions as input to support runtime decisions for selecting efficient CPU frequencies, with the objective of reducing the energy consumption of a HPC system.

*Organization.* The remainder of the paper is organized as follows: the adopted methodology, along with a description of the modeling process, are described in detail in Section 2. In Section 3 we briefly introduce the tools employed for collecting data while Section 4 presents an experimental evaluation of our model. Finally, conclusions and future work are discussed in Section 5.

## 2 Methodology and Modeling

In the following, we propose a conceptual framework to adapt CPU frequency during the execution of an application using hardware metrics. First, we convert the problem to the forecasting of two hardware metrics, namely *CPU power* and *instructions retired*: these are used as an overall power consumption and performance indicator respectively. We then develop a machine learning model to forecast these metrics using the high-granularity sensor data made available by GEOPM and DCDB. The DVFS control mechanism allows to realize a feedback loop, by changing the CPU frequency at runtime, and subsequently making efficient frequency decisions which reduce energy consumption. Figure 1 shows the framework's workflow from feature processing up to training of the model, while Figure 2 outlines how online inference at runtime can be performed.

### 2.1 Overview

We consider the problem of energy minimization for an application with arbitrary execution time, running on a fully-saturated single node. Our domain of control is the CPU and the node level. All other energy consumers can be considered

static or directly proportional to CPU usage, for safe operation of the system. Under such assumption, the total energy consumed by an application is the total power consumed integrated over the execution time. We further simplify the problem by assuming that node-level hardware metrics are enough to sufficiently characterize the application behavior, greatly reducing the dimensionality of data, since DVFS usually operates at the node level.

Supervised learning techniques are well-suited for real-time, online implementations, due to their low computational overhead in training and inference. This work focuses on the training and validation of such a model. Specifically, we explore an *offline* training approach, in which the machine learning model is trained "a priori", and can later be used for online inference of power and instructions. Such a model can also be re-trained online, with recent data, so as to cope with changing system behavior over time. The evaluation of real-time inference against new data is left for future work.

### 2.2 Mathematical Model

We quantify the efficiency of frequency decisions for each point in time based on the following model: first, we assume that a generic application, with total execution time $T$, is divided into N time intervals (also referred to as *lags*) of equal duration $\delta t = \frac{T}{N}$. Such $\delta t$ duration is assumed to be independent from the current CPU frequency setting, and is calculated from a clock source such as NTP. We then define the *power consumption* for each interval $i$ at time $t_i$ as the function $P(f_i, \zeta(t_{i-1}), \zeta(t_{i-2}), ..., \zeta(t_{i-W}))$, where $\zeta(t_k)$ is a vector of hardware metrics for time interval $k$ at time $t_k \in [t_{i-W}, t_{i-1}]$. $W$ is the length of the historical time window used to determine $P$ at time $t_i$, and its value depends on the application and system being analyzed. $f_i$ is the CPU *frequency* to be used at time $t_i$: since frequency decisions are taken at each time interval $i$ using DVFS, affecting the power consumed by the CPU as well as the time taken to progress in the application, observed frequency will also change.

Given the above, in this work we focus on minimizing the energy of each time interval $i$ separately, instead of the application as a whole, neglecting the influence of different decisions on one another. This simplifies the underlying optimization process, at the expense of sub-optimal solutions which do not consider long-term effects. Application throughput at each time interval can potentially decrease when decreasing frequency as the application executes less instructions per unit of time. We approximate this relation by assuming that the throughput of an application in a time interval is inversely proportional to the number of retired instructions. We thus use the latter to compute the energy associated with time interval $i$ by introducing the $Inst$ function, which depends on the same variables as $P$. The minimum energy for a time interval $i$ is then given by:

$$MinEnergy(i) = min_{f_i} \frac{P(f_i, \zeta(t_{i-1}), \zeta(t_{i-2}), ..., \zeta(t_{i-W}))}{Inst(f_i, \zeta(t_{i-1}), \zeta(t_{i-2}), ..., \zeta(t_{i-W}))} \quad (1)$$

Using the mathematical formulation of minimum energy above, we can estimate the most efficient frequency of a time interval $i$ by finding which frequency $f_i$ minimizes the function in Equation 1. This is expressed as the following:

$$f_i^{opt} = argmin_{f_i} \frac{P(f_i, \zeta(t_{i-1}), \zeta(t_{i-2}), ..., \zeta(t_{i-W}))}{Inst(f_i, \zeta(t_{i-1}), \zeta(t_{i-2}), ..., \zeta(t_{i-W}))} \tag{2}$$

Given that frequency lies in a small discrete space of values, online tuning can be performed by evaluating the $P$ and $Inst$ functions for all available frequency values and picking the one which generates the lowest energy value.

### 2.3 Machine Learning Model

We assume the CPU power ($P$) and Instructions retired ($Inst$) for any interval $i$ at time $t_i$ in Equation 1 to be functions of the frequency at time $t_i$, and of the sequence of hardware metrics in the historical time window $[t_{i-1}, t_{i-2}, ..., t_{i-W}]$ of size $W$. After statistical analysis of traces, partial auto-correlation [3] of power and instructions retired indicated that only the most recent 3-5 lags in the sequence are significant. In general, a temporal machine learning model must be used to capture the causal relationships existing in sequence data, which is assumed to be non-independent. Auto-regressive time series models are not suitable as the series is non-stationary. However, instead of using the sequence of time lags as input in our model, we can generate statistical features from the time series of each hardware metric, preserving information content in the temporal dimension. The main benefit is that a simpler supervised learning model can be used, which comes at an expense of generating the features from data before actual inference can be performed. Since our machine learning model should predict the optimum frequency in real-time, the computational effort of generating the features should be as low as possible. Due to this limitation, feature set selection was done prior to evaluation, instead of performing exhaustive generation and "a posteriori" elimination.

The features used in this work are exponentially weighted mean, exponentially weighted gradient, standard deviation, skewness, kurtosis, quantiles(0.25, 0.5 and 0.75), absolute sum of changes and sample entropy. The mean and gradient values are exponentially weighted to give more importance to recent data, which is fundamental for performing regression, as opposed to classification tasks [13]. Extracted features were then employed to train a model for estimating the power consumption and instruction retired metrics for the corresponding set of input features. Considering the non-linearity of the problem, a multi-output random forest regressor was used as learning method. Random forests were chosen due to their robustness against unbalanced, noisy and non-normalized data, and because of their efficient operation in the presence of large feature vectors. Moreover, random forests supply information about the importance of each single feature in the regression process, which is useful for our study.

## 3 Data Collection

In the following we introduce two tools used at LRZ which we employ for data acquisition. Both have a slightly different scope: GEOPM, a runtime system designed for monitoring and optimizing performance and power control at job execution, and DCDB, a continuous monitoring system collecting operational data with complete compute clusters as scope.

### 3.1 GEOPM

The Global Extensible Open Power Manager (GEOPM) is a framework designed to provide scalable abstractions to hardware controls and performance counters for power and energy optimization [4]. The GEOPM runtime executes alongside a regular MPI job to observe application and hardware characteristics, which serve as input for GEOPM's optimization algorithms, called "agents". The operating agent can be selected from a set of readily available or self-implemented plugins for center-specific use-cases. It was crucial for our models to observe the behavior of applications under different frequencies to make correct energy and work load predictions. Thus, for this work we implemented a GEOPM agent that samples hardware metrics every 50ms and changes the frequency to a random available setting every third sampling (150ms). The benchmarks are run with this GEOPM agent multiple times so as to extract characteristics arising from the different frequencies in the same section of an application.

### 3.2 DCDB

LRZ researchers developed DCDB with the objective of providing a holistic solution for fine-grained monitoring of sensor and performance metrics in HPC systems [11]. Support for large cluster deployments is ensured by storing data in a NoSQL wide-column database, while low latency and minimal overhead are achieved by transmitting telemetry data with MQTT messages. DCDB is designed following a plugin-based architecture, with each plugin supporting a specific type of protocol for retrieving data. In the scope of this work, we used DCDB to collect sensor data (through SysFS) and in-band performance metrics (through perfevents [17] and ProcFS[4]). Applications that are executed alongside GEOPM were monitored by DCDB, by sampling each metric every 100ms.

### 3.3 Fitting Two Data Sources

DCDB and GEOPM originally have two different scopes. GEOPM is job-centric, and is able to sample CPU hardware counters at rates of ∼10ms. DCDB's intent is to persistently store time series data of the complete cluster, including hardware counters from compute nodes, but also information from the operating system. Since the tools access different counters via different methods exposed

[4] http://man7.org/linux/man-pages/man5/proc.5.html

by the kernel or directly by the hardware, and at different time resolutions, the respective readings are not synchronized. To be able to utilize both data sets for evaluation of a job execution the excess readings before and after the actual run had to be removed. The sliced time series set of DCDB was then upsampled via linear interpolation to fit the GEOPM sampling rate to complete the dataset. Linear interpolation was also used to fit the DCDB data points to the GEOPM time stamps, and to have consistently-aligned and evenly-spaced data. By using both DCDB and GEOPM, we can assess how useful the combination of different metrics collected from different tools is and if these can be brought together for a specific use-case.

## 4 Experimental Evaluation

In this section we evaluate the performance of our model when predicting the CPU power and instruction retired metrics. We first present our experimental setup and methodology, and then give insights on our results.

### 4.1 Experimental Setup

The collection of data was performed by monitoring different runs of a set of benchmarks from the Coral-2[5] suite on the CooLMUC-3[6] system hosted by LRZ.

*Test Environment.* CooLMUC-3 consists of 148 nodes each equipped with Intel Xeon Phi CPU 7210 processors, operating at frequencies ranging from 1.0GHz to 1.5GHz, 96GB of RAM and Intel OmniPath network interfaces. We designated a single node for data acquisition, to ensure consistency of measurements. Each node runs SUSE Linux Enterprise Server 12 SP3 and comes with Intel performance libraries and compilers, which were used to compile the applications. Intel GEOPM 0.6.0, MPI 17.0.6 and an early version of DCDB were used during our experiments. All models were implemented with the open-source Scikit-learn library for Python.

*Applications.* In order to obtain organic data reflecting the behavior of real HPC workloads and exhibiting different application characteristics, we employed a series of applications from the Coral-2 suite. These are *AMG*, *Kripke*, *LAMMPS*, *Quicksilver* and *Nekbone*. All applications are configured to run with one MPI rank and 62 threads for full node saturation, with two cores reserved for the operating system and the GEOPM controller, and are tuned for an execution time of approximately 8 minutes. For the purpose of this initial model evaluation, and thanks to the diversity of the benchmark programs, we assume these single-node runs to be representative enough of the large-scale runs done at LRZ.

[5] https://asc.llnl.gov/coral-2-benchmarks/
[6] https://www.lrz.de/services/compute/linux-cluster/coolmuc3/

| | GEOPM | GEOPM + DCDB |
|---|---|---|
| Number of features | 81 | 417 |
| Overall training error | 3.9% | 2.4% |
| Overall validation error | 9.1% | 6% |
| Validation error (Power Package) | 3% | 2.2% |
| Validation error (Instruction Retired) | 15.3% | 9.7% |

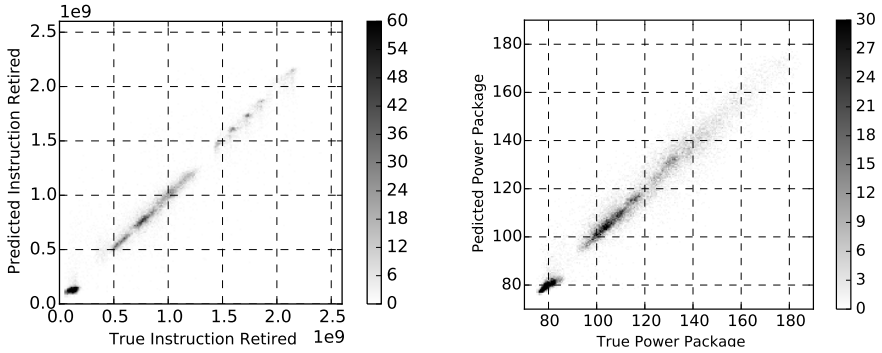
**Table 1.** Summary of the two approaches with the corresponding relative error values.

*Model Configuration.* The random forest regressor chosen for our application was trained using two distinct approaches: first, we train the model using only features from GEOPM data, whereas in the second case we enrich the dataset with DCDB data for further analysis, as discussed in Section 3.3. In both cases, we used data from 5 traces of each application and 5-fold cross validation to test the model's performance, with 53k training samples and 23k validation samples for each combination of folds. We report average results from each pair of training and test sets. When DCDB data is included, the size of input feature vectors to the model was significantly increased. The input statistical features composing the feature vectors at each time step were built using a sliding, overlapping temporal window of the most recent 9 lags, whereas the target CPU power (in the following referred to as *power package*) and instruction retired values are the average of the next 3 lags. We also supply the average CPU frequency of the next 3 lags as an input feature to make the model suitable for the control algorithm described in Section 2.2. The details of the datasets used to train the model with the two approaches described above are outlined in Table 1.

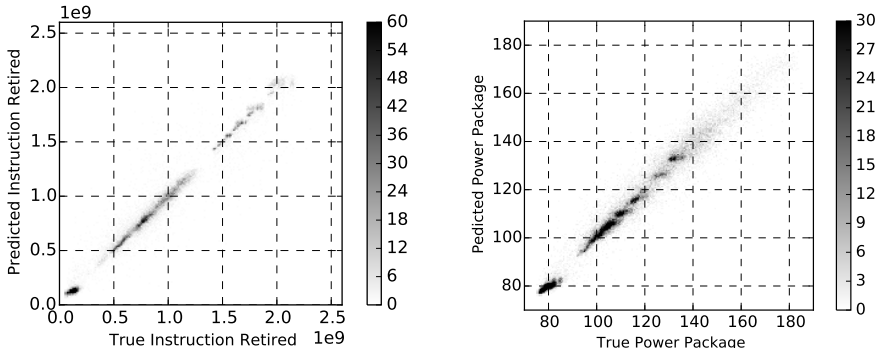
### 4.2 Overview of the Results

Results for regression on acquired data are presented in Figure 3. Specifically, the heatmaps indicate the predicted values from the regressor for instruction retired (left) or power package (right), compared to the actual values, when using only GEOPM data (Figure 3a) or DCDB data as well (Figure 3b). Darker areas indicate higher density of points. In Table 1 we also show the average relative training error and validation error for each separate target as well as combined.

It can be seen that in all cases the results are very positive, with an average relative error lower than 15.3%, and metric values are predicted correctly across the whole range. The power package metric is predicted more accurately than the instruction retired metric, likely because the latter corresponds to an inherently noisier sequence, and is influenced by unpredictable factors such as OS interference. Moreover, adding DCDB data leads to slightly more precise prediction. This last result is more pronounced when observing the average relative error for each target, which is shown in Table 1. For comparative purposes we also implemented a *baseline* predictor, which uses the average of the latest three lags as prediction for the power package and instruction retired metrics. Using




(a) GEOPM data.




(b) GEOPM+DCDB data.

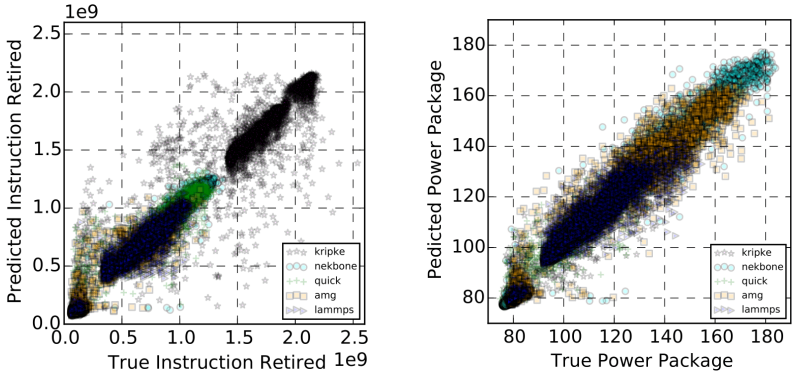
**Fig. 3.** Model performance in predicting the instruction retired (left) and power package (right) with different data sources.

this naive implementation, we observed an average relative error of 23.7% and 168.6% respectively, proving the intrinsic complexity of this regression problem.

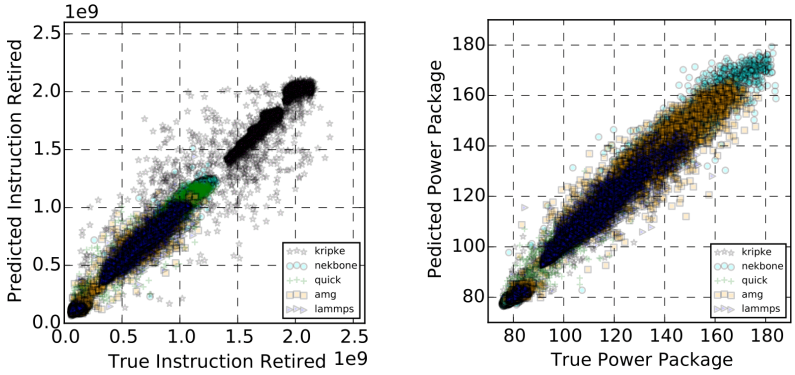
### 4.3 Per-Application Analysis

Here we show regression results for each application in the dataset separately, so as to expose artifacts and effects that may be associated to their behavior. Results are shown in Figure 4: the scatter plots depict predicted values against the actual ones, like in Figure 3, and are color-coded for each application.

The results reflect those discussed in Section 4.2. However, some interesting effects can be observed: the LAMMPS and AMG applications, for example, show two distinct operational behaviors, which translate into two separate clusters visible in the scatter plots. This behavior is successfully captured by our model. It can also be seen that the Kripke application shows a comparatively higher




(a) GEOPM data.




(b) GEOPM+DCDB data.

**Fig. 4.** Model performance in predicting the instruction retired and power package for each benchmark.

spread in the predicted instruction retired values, which is mitigated when using both GEOPM and DCDB data. However, it can be seen that overall performance for our model is equally good for all applications, implying that it is generic enough to characterize the diversity of HPC work loads.

### 4.4 Evaluation of Feature Importance

As mentioned in Section 2.3, random forest regressors are capable of extracting the most dominant features for generating the model. Table 2 indicates the most important features in training the model with the corresponding weight, when using only GEOPM data (left) and DCDB data as well (right).

As it can be seen, the most important feature in both cases is associated to past instruction retired observations, which is expected. The average CPU frequency of the next 3 lags (*frequency*) also plays an important role, proving the validity of the model discussed in Section 2.2. When using GEOPM data alone,

| GEOPM | | GEOPM+DCDB | |
|---|---|---|---|
| **Score** | **Name** | **Score** | **Name** |
| 0.208 | geopm inst-retired mean exp weighted | 0.376 | geopm inst-retired mean exp weighted |
| 0.171 | geopm cycles thread kurtosis | 0.144 | dcdb hfi0temp grad exp weighted |
| 0.071 | geopm cycles reference quantile 0.25 | 0.121 | dcdb col idle grad exp weighted |
| 0.060 | geopm frequency | 0.098 | dcdb hfi0temp diff sum |
| 0.048 | geopm energy dram quantile 0.25 | 0.055 | dcdb references quantile 0.5 |
| 0.047 | geopm energy pkg quantile 0.75 | 0.052 | dcdb energy quantile 0.75 |
| 0.045 | geopm power pkg quantile 0.75 | 0.042 | dcdb hfi1temp grad exp weighted |
| 0.044 | geopm power pkg quantile 0.5 | 0.040 | dcdb intr quantile 0.25 |
| 0.040 | geopm power pkg kurtosis | 0.022 | dcdb col idle diff sum |
| 0.038 | geopm inst-retired quantile 0.5 | 0.014 | geopm frequency |

**Table 2.** Most important features as quantified by a random forest regressor.

the remaining most important features are mostly related to past observations of the power package and energy metrics. When using DCDB data as well, the set of important features is more heterogeneous: metrics indicating time spent by the CPU in idle (*col idle*) operation can be seen, as well as temperature sensors (*hfi0temp* and *hfi1temp*) and interrupt counters (*intr*). This, coupled with the results discussed in Section 4.2, shows the importance of using diverse monitoring data for our regression problem.

## 5 Conclusions

In this paper we developed a machine learning model to predict the CPU power and instruction retired metrics, designed to support efficient frequency decision during the execution of HPC applications. We first collected data by running a set of applications on a production HPC system using the DCDB and GEOPM frameworks, which was later combined. We then derived a mathematical model to predict CPU power draw and instructions retired, based on the hardware metrics available, and thus make appropriate frequency decisions to minimize energy consumption. The model is suitable for online training and inference. For this study, we evaluated the generated model offline using training data obtained on LRZ's CooLMUC-3 system, using various applications from the Coral-2 suite.

The results show that the model is generic, with high accuracy in both predicted power as well as instructions retired across all applications. Moreover, we show the effectiveness of combining data from cluster monitoring and high-fidelity runtime systems: the monitoring system has access to data not accessible by the runtime and, on the other hand, the runtime system brings a scalable infrastructure to implement agents for online optimization. As future work, we plan to evaluate our model with data from multiple HPC nodes, and further test its effectiveness by implementing an agent for online frequency tuning using predictions from new data as captured in real time.

*Acknowledgements.* This work originated from the TUM Data Innovation Lab, and was further supported by Intel Deutschland GmbH and LRZ.

## References


1. Agelastos, A., Allan, B., Brandt, J., Cassella, P., et al.: The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proc. of SC 2014. pp. 154–165 (2014)
2. Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., et al.: A case study of energy aware scheduling on supermuc. In: Proc. of ISC 2014. pp. 394–409. Springer-Verlag (2014)
3. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time series analysis, forecasting and control (4th ed.). In: Hoboken, NJ: Wiley. p. Chapter 3.2 (2008)
4. Eastep, J., Sylvester, S., Cantalupo, C., Geltz, B., Ardanaz, F., et al.: Global extensible open power manager: A vehicle for HPC community collaboration on co-designed energy management solutions. In: Proc. of ISC 2017. pp. 394–412 (2017)
5. Jones, N.: How to stop data centres from gobbling up the worlds electricity. Nature **561**, 163–166 (2018)
6. Koomey, J.G.: Worldwide electricity used in data centers. Environmental Research Letters **3**(3), 034008 (jul 2008)
7. Koomey, J.G.: Growth in data center electricity use 2005 to 2010. Analytics Press (August 2011), `http://www.analyticspress.com/datacenters.html`
8. Kumar, A.S., Mazumdar, S.: Forecasting hpc workload using arma models and ssa. In: Proc. of ICIT 2016. pp. 294–297 (2016)
9. Kunkel, J., Dolz, M.F.: Understanding hardware and software metrics with respect to power consumption. Sustainable Computing: Informatics and Systems **17**, 43–54 (2018)
10. Lin, X., Wang, Y., Pedram, M.: A reinforcement learning-based power management framework for green computing data centers. In: Proc. of IC2E 2016. pp. 135–138. IEEE (2016)
11. Netti, A., Mueller, M., Auweter, A., Guillen, C., et al.: From facility to application sensor data: Modular, continuous and holistic monitoring with DCDB. In: Proc. of SC 2019. ACM (2019)
12. Triki, M., Wang, Y., Ammari, A., Pedram, M.: Hierarchical power management of a system with autonomously power-managed components using reinforcement learning. Integr. VLSI J. **48**(C), 10–20 (2015)
13. Tuncer, O., Ates, E., Zhang, Y., Turk, A., et al.: Online diagnosis of performance variation in HPC systems using machine learning. IEEE Transactions on Parallel and Distributed Systems (2018)
14. Wang, B., Terboven, C., Mller, M.S.: Performance Prediction under Power Capping. In: Proc. of HPCS 2018. pp. 308–313. IEEE (2018)
15. Wang, Y., Xie, Q., Ammari, A., Pedram, M.: Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. In: Proc. of DAC 2011. pp. 41–46 (2011)
16. Wang, Z., Tian, Z., Xu, J., Maeda, R.K.V., Li, H., et al.: Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system. In: Proc. of ASP-DAC 2017. pp. 684–689. IEEE (2017)
17. Weaver, V.M.: Linux perf_event features and overhead. In: Proc. of the FastPath Workshop 2013. vol. 13 (2013)
18. Wilde, T., Auweter, A., Shoukourian, H.: The 4 pillar framework for energy efficient HPC data centers. Computer Science - R&D **29**(3-4), 241–251 (2014)
19. Yang, S., Shafik, R.A., Merrett, G.V., Stott, E., Levine, J.M., et al.: Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In: Proc. of the PATMOS Workshop 2015 (2015)