Recognizing HPC Workloads Based on Power Draw Signatures

Sven Köhler[1], Lukas Wenzel[1], Max Plauth[1],
Pawel Böning[2], Philipp Gampe[2], Leonard Geier[2], and Andreas Polze[1]
Hasso Plattner Institute for Digital Engineering
Operating Systems and Middleware Group
University of Potsdam, Germany
[1] {firstname.lastname}@hpi.uni-potsdam.de
[2] {firstname.lastname}@student.hpi.uni-potsdam.de

***Abstract*—The power draw of computing infrastructure—besides being a critical operating resource—can give valuable insights into the type and behavior of workloads running on it. In consequence, runtime power analysis can be a promising non-invasive monitoring approach. Recent work has shown that a system's power draw can support reliable conclusions about running workloads, which serves as a basis for runtime placement decisions to adapt the system's cumulative energy demand to the available energy supply in a volatile electricity grid.**

In this work, we reproduce earlier findings on the classification of running workload from a set of previously known workloads purely through external power measurements. Using a k-nearest neighbors classifier, we identify workloads of the NAS benchmark suite with a macro F1-score of 98% for OpenMP-based implementations and 85% for MPI-based implementations.

***Index Terms*—Power and Energy Measurements, Heterogeneous Systems, Machine Learning, System Software**

I. Introduction

The power supply is a critical operating resource for all types of computing systems, ranging from embedded devices over desktop systems to high-performance supercomputers. In consequence, understanding the power characteristics of the executed workloads is of supreme importance when planning and later operating a particular system. These characteristics encompass both the peak power demand as well as the cumulative energy demand over time. The upper limit to the power draw is not only imposed by each of the system's components' power ratings—the operational costs of the power supply itself can be subject to dynamic changes due to the increasing share of renewable energy sources in the power grid [1].

With a given set of workloads with known characteristics to be executed, a computing system's job manager has three options to adapt the power and energy demand to the currently available resources: Firstly, it can *work differently* by changing, for example, the clock frequency and supply voltage of the processing units. Secondly, the computing system can *work another time*, by deferring or picking a particular workload that matches the current supply and cost. And lastly, the system can *work elsewhere* by moving a workload between heterogeneous processing units (i.e., accelerators) [2].

To make decisions based on any of those three approaches, a job manager requires information about the power draw profile for each workload in each of its program phases. These profiles can be retrieved from *a priori* classification, or they can be detected *on-the-fly* at runtime if no such classification is provided beforehand with the job. While workload classification can be achieved by static means like code analysis [3] or dynamic features like performance metrics [4], [5] at runtime, the system's power draw itself can be a promising indicator of the power profiles of running workloads, as it can be ascertained using non-invasive measurement facilities that can be completely external to the observed system.

The observable information in the time-series of a system's power draws is only an aggregate of a significantly more complex system state, where effects on different layers of abstraction overlap. On the hardware level, different units in the system like CPUs, memory, accelerators, and interconnects exhibit individual power draws. These are ultimately a response to workload behavior, but advanced optimization strategies such as caching or out-of-order execution introduce a certain level of diversion, precluding a direct mapping between executed operations and each unit's power draw. On the workload level, multiple independent workloads as well as background processes can be executed concurrently and are likely to interfere. Thus the combined power draw is not necessarily the same as the sum of individual workload power draws measured in isolation. Furthermore, workloads may exhibit different behavior and consequently different power draws in the course of different execution phases.

Taken together, these characteristics render approaches that explicitly derive the system state from the power draw time-series using fixed decoding rules impractical. Also, such a set of rules would necessarily be specific to a particular hardware platform. In contrast, an automatic classification mechanism can be adapted to a particular system setting during a training phase, and may be expected to produce more accurate results in a broader range of situations.

Therefore, this work explores the feasibility of a system that classifies a running workload from a set of previously known and analyzed workload characteristics purely through external power measurements of the observed system using a k-nearest neighbors classifier with manual hyperparameter tuning. We expect workloads of similar power draw characteristics to

respond similarly to placement decisions, as caused by the utilization of different system resources. Hence, the membership in a particular characteristics class would be a good indicator for the placement strategy choice by a job manager. In our approach, the automatic classifier is trained to classify a single running workload according to a predefined set of placement classes, which are coupled with distinct sets of rules for the job manager on how, where, and when to execute this workload. After an initial recognition phase, a job manager can use this classification to optimize the placement of the workload among other workloads in the system. A complete system must theoretically also involve on-going monitoring as the characteristics may change with the program phases and the placement must be updated.

In particular, our approach builds up on top of a pre-print by Copos and Peisert [6], who investigate the feasibility of detecting abuse of large-scale HPC computing resources based on the overall power consumption of multi-node compute racks. Like the authors, we limit our scope to recognizing a single workload running at a time due to the frequently encountered batch execution of HPC jobs. Though we are not interested in the binary classification of legitimate and illegitimate workloads, but rather a distinction between multiple workload classes sharing similar placement and scheduling guidelines, their results are still promising: From a set of known legitimate workloads, they train a classifier based on random forests achieving up to 97% precision and 95% recall, which would be more than sufficient for our approach, where the results are used for optimized placement of unknown workloads and have no consequences for correctness. Nevertheless, as the only basis for the classification are the power draws and placement classes regard workloads as equal up to their power draw characteristics, their approach is likely applicable.

One important constraint on the classification mechanism is that it needs to be lightweight and reproducible. The energy cost of each classification as well as the amortized cost of training and adaptation for new jobs may not exceed the potential energy savings gained using a–yet to be investigated—optimized placement strategy. Current deep learning methods, which require multiple network layers and considerable amounts of training data (hence repeated runs of the test workload) are prone to high energy demands themselves [7], [8] and are thus not taken into closer consideration at this early point in our scenario. Whether these assumptions hold true is left to future analysis.

II. Related Work

Identifying workloads based on non-intrusive side-channel information—like power signatures in particular—is of ongoing interest, ranging from mobile consumer devices [9] to HPC systems [10], [11].

The techniques are commonly split up into two distinct tasks: *data collection* and *data analysis/transformation* [11], [12]. Both are adjusted to the context in which they are going to be used. For instance, during the collection step, systems for energy-constrained environments (e.g. mobile) such as the ones developed by Jacoby et al. [13] and Kim et al. [12] don't yield data for the entire runtime of a workload, but only for periods of elevated power consumption that are considered of particular interest. In a less constrained environment, it is entirely possible to record and store complete power traces [11], [6], [9], [10].

Data collection is typically followed by a data transformation step to aggregate statistical features. Which features are most suited to a given context varies with the hard- and software configuration [11]—a set of features that delivers good results for a mobile device does not necessarily do for an HPC cluster. Interestingly enough, it seems like a small number of features already allows the workload differentiation in some cases. With HPC clusters, for instance, a feature vector of the normalized maximum and the normalized median (the respective value divided by the minimum value) results in good clusterings of workloads for some authors [10], while aforementioned Copos and Peisert [6] use larger feature vectors as signatures. On an Intel Xeon-based rack, they aim to detect *anomalies*—undesired workloads like mining electronic currency—among other workloads considered *benign*.

In this work we re-evaluate the approach of Copos and Peisert on an IBM OpenPower machine with the particular objective to distinguish workloads for later placement decisions rather than detecting an anomalous workload.

III. Experimental Design

This section introduces details of our experiment's design. In addition to detailed description of the employed *hardware platform*, the workflows employed for *data collection*, *feature extraction*, and *classification* are documented hereinafter.

A. Hardware Platform

Our workload was executed on an *IBM Power System S824L* [14], which features two POWER8 CPUs with ten cores, each configured with a Simultaneous Multithreading (SMT) level of eight hardware threads per core. For this paper, we limit our experiment to CPU-based workloads despite available accelerator cards in this test machine and will not yet investigate the energy impact of workload placement decisions in a heterogeneous system.

The S824L platform provides system-wide power readings through the Intelligent Platform Management Interface (IPMI), which is accessible from within the system for example using Linux' *hwmon sysfs interface*. However, we found the reported values too course-grained and thus not feasible for any analysis, although on a newer generation IBM AC922—unavailable for our experiments—these values looked more promising. More reliable power measurement counters on the OpenPower platform are available via its power and thermal management unit, the On-Chip-Controller (OCC) [15]. Accessing those counters typically requires a custom-build firmware. We ruled out this approach, as our lab's computing platform is a shared resource and other experiments relied on reproducible environments.

Instead, we opted for external measurement devices, that intercept the system's power supply and measure the power draw from the wall socket. We employed two *Microchip MCP39F511N* [16] power meters, each allowing for two channels up to 15 A at a maximum of 230 V, which is sufficient for the four power supply units of the S824L system. An MCP39F511N measures the power demand over a 2 mΩ shunt with a 24-bit delta-sigma analog-to-digital converter (ADC). This allows an accuracy of 0.5 %. The sampling rate is phase-locked to the line frequency (50 Hz in our country) and allows a configurable number of samples per line-frequency cycle.

B. Data Collection

In order to reduce self-interference on the S824L test machine for the *data collection* workflow, we spatially and temporally separated the workload execution from the measurement and analysis. The USB data ports of the two MCP39F511N's are attached to an external measurement machine. This is comparable to a future system where the functionality is implemented, for example, in the OCC communicating with the operating system using firmware calls.

On the measurement machine, the power samples are captured using PINPOINT [17], a tool that can report the cumulative energy draw as well as the power time-series for an executed process. File exchange, communication, and synchronization between the test and the recording machine are orchestrated using a reusable SSH connection and a custom build controller—implemented as a simple HTTP app. For each workload the executable is transferred to the S824L, started and after termination, the continuously recorded power draw series is trimmed to the execution timestamps as reported by the controller before being stored for further processing. All recordings were taken with minimal other active processes to reduce any noise in the training data.

C. Feature Extraction

The feature extraction workflow outlined in Figure 1 is applied in this work and is comprised of the four stages *sensor fusion*, *segmentation*, *segment grouping* and *feature calculation*, which are explained in further detail hereinafter.

1) Sensor Fusion: Using a dedicated channel for each of the four power supply units of the employed S824L system, each power draw reading in our setup is a four-dimensional data vector. To reduce the four-dimensional vector to a scalar value, the *sensor fusion* stage applies one of the four simple reduction functions *sum*, *average*, *min*, and *max*. Significantly reduced processing times in later stages are the main motivation for reducing the four-dimensional vectors to scalar values. The reduction is performed under the assumption that all power supply units feed the same internal voltage rails and therefore the individual load on each power supply should not provide any additional information, e.g., on what particular system components may be under load at any time.

2) Segmentation: The continuous time series of fused power draw samples needs to be split into *segments* (sometimes also known as *windows*) for further processing. Most

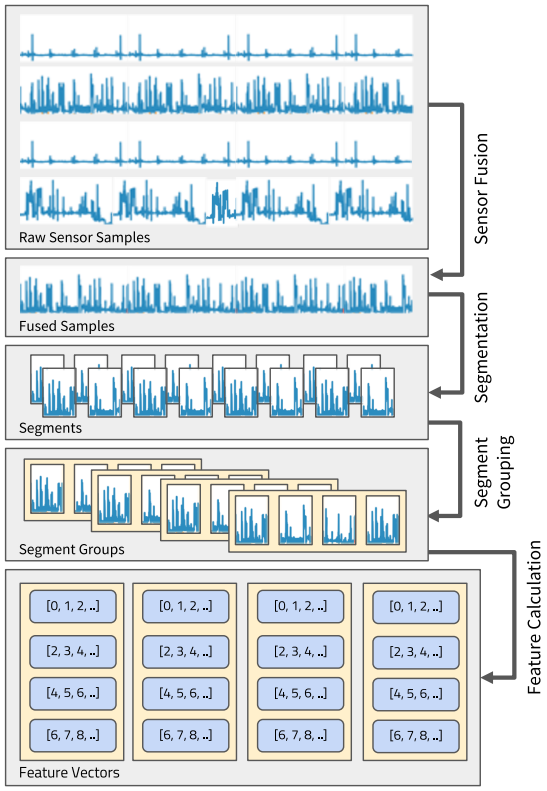


Fig. 1. Workflow diagram of the feature extraction. The four feature extraction stages are denoted on the right. The labels on the left stand for the type of data being processed in each stage.

notable, the use of segmentation is crucial to enable *on-the-fly* classification at runtime based on the continuous power draw measurements. Furthermore, employing segments of consistent size and value domain enables a broader range of classification algorithms that can be applied to the data. Finally, dividing the continuous time-series measured for a workload into short individual segments enables us to capture different execution phases. These phases can differ widely in their behavior and thus might confuse a classification algorithm when applied to the entire time series. This method leaves room for potential workload-phase detection but also for determining the similarities of workloads during different stages.

Segments have two parameters: *size* and *stride*. The *size* defines how many continuous samples a segment is comprised of, whereas the *stride* defines the offset of a segment relative to its predecessor. Segments can overlap or have gaps between them if the *stride* is chosen accordingly. The parameters need to be tuned to group relevant information for a stretch of time.

The *feature calculation* stage can only capture patterns with a period smaller than the segment *size*, so this parameter should be large enough in order not to obscure potentially relevant patterns. Choosing a too large segment *size* may exceed the expressive capacity of the feature vector the segment is condensed into during the *feature calculation* stage. For an excessively large *stride*, large gaps between segments and a small number thereof are unlikely to facilitate adequate

classification. If the *stride* is too small, a lot of redundant segments are generated, resulting in unreasonable resource demand and processing times.

3) Segment Grouping: A potential disadvantage of calculating features solely based on single segments is that recurring power draw patterns stretching across multiple segments are potentially not captured by this approach. As a countermeasure, neighboring segments are combined into *segment groups* to capture temporal locality across a longer stretch of time. The *segment grouping* stage groups segments based on the parameters *group size* and a *group stride*, analog to the parameters used to generate segments. Similarly, both of these parameters require careful tuning.

4) Feature Calculation: Based on the segment groups, the *feature calculation* stage produces a feature vector for each segment group, using relatively simple operations to restrict the computational effort to a reasonable amount. The feature vector is comprised of *time-domain statistics* such as *absolute energy*, *minimum*, *maximum*, *mean*, *standard deviation*, and *quantiles*. Furthermore, the *power-spectral density* is determined in the frequency-domain, yielding the strength of the variations as a function of frequency. The three strongest frequencies identified by this analysis as well as their amplitudes are also part of the feature vector.

D. Classification

The classification workflow consists of the two stages *scaling* and *classification* that the feature vectors have to go through. While the *scaling* stage is optional, its omission is very likely to negatively impact the accuracy of the *classification* stage. Both stages are implemented using the *scikit-learn* [18] library. After the feature extraction stage is finished, the classification stage can use the resulting feature vectors for training, validation, or prediction. The classification workflow can be operated in one of the three modes *training/testing*, *validation* and *prediction*.

1) Scaling Algorithm: Depending on the choice of the scaling algorithm, each set of feature vectors is scaled to one of the ranges $[0, 1]$ or $[-1, 1]$ as many classifiers are designed with the assumption that values vary on comparable scales. Our classification workflow can be configured to used one of the scaling algorithms *MinMaxScaler*, *StandardScaler*, and *RobustScaler*. The first two scaling algorithms require no particular parameter adjustment and are particularly sensitive to outliers which can be a desired feature to expose outliers more strongly. The *RobustScaler* algorithm reduces outliers by scaling the data according to a percentile range.

2) Classification Algorithm: Similarly to the scaler, the selection of the classification algorithm can be considered as the main parameter of this stage. The available classification algorithms are *LogisticRegression*, *DecisionTree*, *LinearDiscriminantAnalysis*, *GaussianNB*, *SupportVector*, and *KNearestNeighbor*.

3) Modes of Operation: In *training/testing* mode, a model is created based on labeled feature vectors, with the labels identifying the workload under test. The *validation* mode is used to validate a previously trained model in a separate step, e.g. for fine-tuning of meta-parameters. In addition to the model generated by the *training/testing* mode, the *validation* mode also operates on labeled feature vectors. Finally, the *prediction* mode uses the trained model yielded by the *training/testing* mode to assign unlabeled feature vectors to one of the trained workloads.

TABLE I
Accuracy and execution time for different fusion functions. The one employed by us highlighted in italics.

Fusion Function	Run-time	Avg. Accuracy
None	95.04 s	95.45 %
Sum	*28.34 s*	*95.44 %*
Average	27.56 s	95.51 %
Minimum	30.69 s	95.49 %
Maximum	31.60 s	95.43 %

IV. Parameter Tuning

In this section, we present how the meta-parameters are tuned for evaluation. As mentioned in section III, the workflows for *data collection*, *feature extraction*, and *classification* require a certain degree of configuration. Choosing the optimal configuration parameters is not trivial and requires experience and testing, as *meta-parameters* can have a significant impact on the quality of the results. As these *meta-parameters* need to be set manually, this section explains how the *meta-parameters* were chosen for the evaluation of our work.

A. Data Collection

For the workloads under test, the problem size for each workload has been dimensioned to result in execution times of roughly five minutes in order to yield sufficiently large power draw recordings of comparable lengths. Furthermore, the power draw recordings were performed for ten repeated executions, allowing us to eliminate sporadic behavior during training and validation. We configured our power meters to a sampling rate of $200\,\mathrm{Hz}$.

B. Feature Extraction

Trial and testing were done to determine a configuration for the *meta-parameters* of the *feature extraction* workflow that result in an adequate accuracy of the *classification* step. As denoted in Table I, the choice of the reduction operator employed during the *sensor fusion* stage only has a marginal impact on the accuracy. Not performing sensor fusion at all does not have a significant impact on the classification accuracy, but results in much longer processing times. For the evaluation, the *sum* operator is used—reflecting our assumption that all power supply units are feeding into the same power rails.

To identify decent values for the *segmentation* parameters *segment size*, *segment stride*, *group size*, and *group stride*, various configurations and their respective accuracy were tested as illustrated in Figure 2. As the accuracy is subject to variance due to the randomized train/test sets, all configurations yielding accuracy scores below 95% were eliminated, resulting

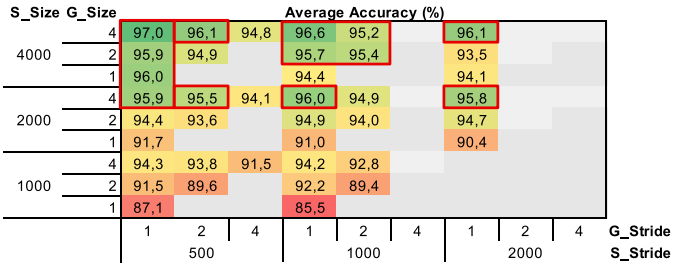


Fig. 2. Accuracies in the NAS-OMP experiment averaged across all available classifiers (higher is better). From the various configurations for size/stride of segments/groups, first all configurations yielding accuracies above 95% were identified as potential configuration candidates in a first step (highlighted with red border). Configurations filtered due to low accuracy or a small number of resulting feature vectors are shown in light grey. Cells in dark gray correspond to invalid parameter configurations, i.e., where the stride exceeds the segment/group size.

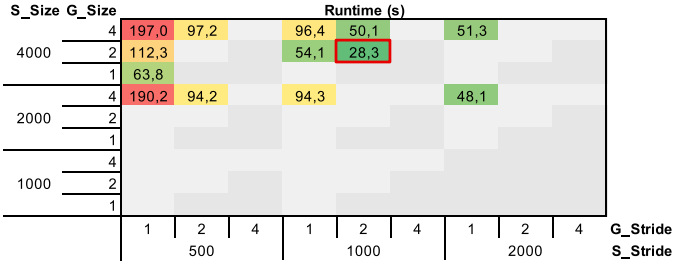


Fig. 3. Required runtime of the feature extraction and classification (lower is better) of the configurations chosen above. In a second step, the size/stride of segments/groups yielding the shortest execution time was selected, with the prospective usage as live classification during run-time in mind. The runtime shown is the average of 20 measurements corresponding to 5 minutes of workload execution.

in the final configuration candidates shown in Figure 3. With only minor differences in their accuracy scores, execution time was used as the deciding factor for picking the parameter configuration documented in Table IV for our evaluation.

C. Classification

The selection of meta-parameters for the *classification* workflow determines how the system organizes feature vectors into categories and how well it behaves in the presence of edge cases, outliers, and anomalies. As each scaling algorithm and classification algorithm have particular strengths and weaknesses regarding different types of feature distributions, it can be crucial to select them carefully.

To identify the scaling algorithm yielding the highest accuracy scores for our scenario, the *Standard*, *MinMax*, and *Robust* scaling algorithms as well as the omission of the *scaling* stage were tested (see Table II). Omitting the *scaling* stage results in a significantly lower accuracy compared to configurations that use a scaling algorithm. Even though both the *MinMax* and the *Robust* scaling algorithm provide similarly high levels of accuracy, we have decided to use the *MinMax* scaling algorithm in our evaluation as it is very sensitive to outliers which could help expose anomalies.

Lastly, the classification algorithms *LogisticRegression*, *DecisionTree*, *LinearDiscriminantAnalysis*, *GaussianNB*, *SupportVector*, and *KNearestNeighbor* were tested. The data documented in Table III reveals that k-nearest neighbor achieves the highest accuracy in our experiments.

TABLE II
Accuracy of different feature scaling algorithms.
The one employed by us highlighted in italics.

Scaling Algorithm	Avg. Accuracy
No Scaling	69.50 %
Standard	92.62 %
MinMax	*95.44 %*
Robust	95.40 %

TABLE III
Accuracy of different classification algorithms.
The one employed by us highlighted in italics.

Classification Algorithm	Avg. Accuracy
Logistic Regression	95.67%
Decision Tree	94.63%
Linear Discriminant Analysis	97.36%
Gaussian Naive Bayes	92.37%
Support Vector	96.42%
K Nearest Neighbor	*97.93%*

The final summary of all *meta-parameter* that were identified in this section are denoted in Table IV.

V. Evaluation

In this section, several experiments are conducted based on the experimental design detailed in section III. The goal of these experiments is to evaluate what accuracy our classification approach can achieve for the task of identifying HPC workloads running on our test system.

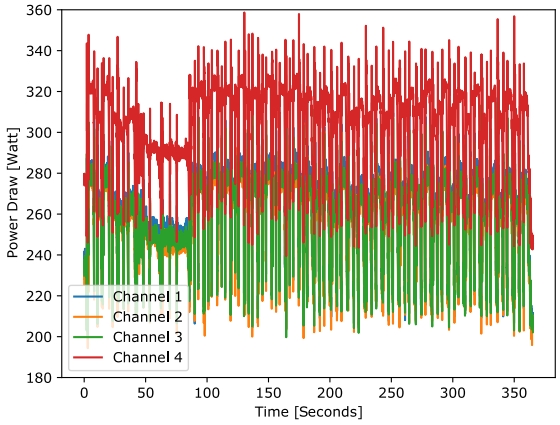
A. Workloads

With future placement decisions based on specific workload characteristics in mind, good workloads for our experiments should serve as instances of particular known problem classes like, e.g., the *Berkeley Dwarfs* [19]. Yet, due to implementation differences workloads of the same class are likely to yield different power signatures, just as with other performance metrics. For our experiments, we thus first want to discriminate between workloads of the same implementation strategy.

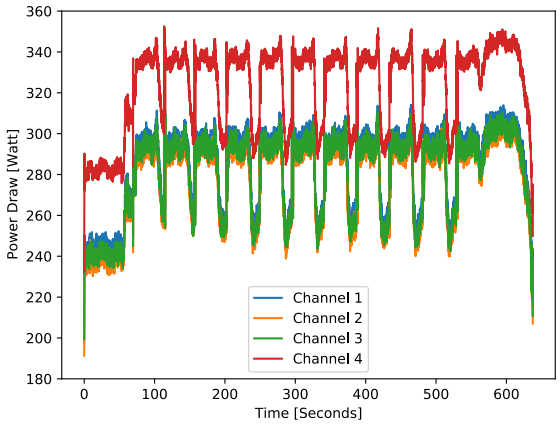
The *NAS Parallel Benchmark* (NPB) suite [20]—which in turn influenced the definition of the Berkeley Dwarfs—

TABLE IV
Final selection of meta-parameters.

Parameter	Value
Segment Size	4000 ms
Segment Stride	1000 ms
Group Size	2
Group Stride	2
Fusion Function	Sum
Scaling Alg.	MinMax Scaler
Classification Alg.	K Nearest Neighbor (k = 1)



(a) Power draw series of the *Fourier transform* benchmark



(b) Power draw series of the *Embarrassingly parallel* benchmark

Fig. 4. Two exemplary four-channel power time-series as recorded by the two Microchip MCP39F511N attached to our IBM S824L test machine. They show the changing power draw signatures from the start of each workload to its completion. No sensor fusion has been applied so far on this raw input data. Note, that the y-axis is trimmed for better readability.

provides multiple implementation strategies for the same problem. We choose to investigate those using MPI and OpenMP, as of version 3.4.1 of the NPB suite.

Each benchmark of the suite can be configured to use different problem sizes that mainly affect execution time and memory consumption. All OpenMP-based benchmarks used 160 hardware threads, or 64 MPI processes respectively. We have selected problem size classes resulting in an execution time of roughly five minutes (or 6000 samples) in order to yield sufficiently large power draw recordings of comparable lengths. Each recording was repeated ten times. Two examples of the captured power time series can be found in Figure 4.

B. Experiments

For these experiments, the recorded data is always split into a *train set* and a *test set*. Both sets consist of labeled files that contain the power draw recordings of the measured workloads. In this case, each file represents a separate run of a workload. The *train set* is processed by the *feature extraction* workflow, and the resulting feature vectors are randomly shuffled and

	Idle	bt.D	cg.D	ep.D	ft.D	is.D	lu.D
Idle	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
bt.D	0.00%	99.24%	0.00%	0.00%	0.76%	0.00%	0.00%
cg.D	0.00%	0.00%	99.84%	0.00%	0.16%	0.00%	0.00%
ep.D	0.00%	0.00%	0.00%	93.62%	6.38%	0.00%	0.00%
ft.D	0.00%	2.59%	0.29%	2.30%	93.68%	0.29%	0.29%
is.D	0.00%	0.00%	8.70%	0.00%	0.00%	86.96%	4.35%
lu.D	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%

Fig. 5. Confusion matrix of experiment 1 (NAS-OMP distinction). Each cell corresponds to how likely the two workloads can be confused with each other (macro F1-score of 98%). The labels combine the NAS benchmark's name (e.g. BT for block tri-diagonal solver) and the problem size class (e.g. D for the third-biggest problem size). The high recall rates show that a distinction between different benchmarks of similar problem sizes is possible.

	Idle	bt.D	cg.D	ep.D	ft.D	is.D	lu.D
Idle	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
bt.D	0.00%	97.69%	0.00%	0.00%	2.31%	0.00%	0.00%
cg.D	0.00%	0.00%	87.93%	0.18%	11.89%	0.00%	0.00%
ep.D	0.00%	0.00%	0.97%	69.07%	0.81%	29.15%	0.00%
ft.D	0.00%	3.94%	0.41%	1.04%	93.98%	0.62%	0.00%
is.D	0.00%	0.00%	0.16%	51.03%	0.79%	48.01%	0.00%
lu.D	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%

Fig. 6. Confusion matrix of experiment 2 (NAS-MPI distinction). The entire experiment has a macro F1-score of 84%. Although a distinction between some workloads is possible, especially the integer sort (IS) and embarrassingly parallel (EP) benchmarks suffer from a high confusion rate. Further experiments indicate that the confusion rate can be reduced by limiting the number of threads for this particular case.

fed into the *classification* workflow. During classification, a portion of the *train set* (default: 25%) is used to validate the model immediately after training. This step was primarily used for *meta-parameter tuning* as it avoids over-fitting of the *meta-parameters* on the final *test set*. During the evaluation, this validation step can be treated as a minor indicator of success—the final evaluation is done on the *test set*.

In each experiment, we perform six repeated runs of each benchmark, where each recording is labeled with the benchmark executed during this particular run. Four recordings are used for the *train set* (25% validation), whereas the remaining two are used for the *test set*.

a) Experiment 1: Distinguishing OpenMP benchmarks: For the first experiment, the OpenMP-based implementations of the *bt*, *cg*, *ep*, *ft*, *is*, and *lu* benchmarks from the NPB suite have been employed. This experiment tests if parallel workloads with different computational tasks can be distinguished based on the resulting power draw. In this experiment, no variation was introduced across the workloads regarding problem size, parallelization, or implementation. The results of the first experiment are documented in Figure 5.

b) Experiment 2: Distinguishing MPI benchmarks across varying levels of parallelism: For the second experiment, the MPI-based implementations of the *bt*, *cg*, *ep*, *ft*, *is*, and *lu* benchmarks from the NPB suite have been employed. In contrast to the first experiment, the benchmarks were executed with varying thread counts. The goal of this experiment is to test if specific workloads can be identified even across varying degrees of parallelization. The results of the second experiment are documented in Figure 6.

C. Discussion

The results of our first experiment (see Figure 5) confirm that the approach presented in this paper can yield high accuracy for the task of distinguishing parallel workloads executed on the IBM S824L test system. With a macro F1-score of 98%, the first experiment demonstrates that performing workload classification solely based on power draw is generally feasible for a well-known set of workloads.

Varying thread counts in the second experiment still yielded decent accuracy levels for distinguishing different workloads (see Figure 6). To verify if the classification errors might be caused by the varying thread counts, additional tests have yielded better accuracy when separate labels were used to denote runs of the same benchmark but with different thread counts (data not shown).

VI. Conclusion

In order to build a job scheduler that can improve the energy efficiency of computing resources by placing workloads on the hardware component most suited for the task at hand, an unintrusive, low-overhead mechanism for attributing workloads to a certain class of characteristics is vital. As a first step towards realizing this vision, we have evaluated the feasibility of such an automated classification component in this work by adapting the approach of Copos and Peisert [6], who have suggested the use of power draw signatures to detect anomalies on a system. Instead of performing anomaly detection, we have demonstrated that our approach can be used to identify different benchmarks of the *NAS Parallel Benchmark* (NPB) suite based solely on power draw measurement series, achieving a macro F1-score of 98% for OpenMP-based benchmarks and 85% for MPI-based benchmarks.

As we investigated these two implementation strategies separately, the question remains if workloads of the same problem class but using different strategies should be considered as one group or not. A preliminary experiment found that using our approach we could distinguish OpenMP-based NAS benchmarks from MPI-based with a macro F1-score of 88%. Whether this has to be accounted to overfitting to a specific implementation or implies a more general distinction needs further investigation. Likewise, the impact of noise or multiple workloads running simultaneously on the same machine is left for future work.

Even though this paper is focused on identifying benchmarks based on pre-recorded power draw series, preliminary experiments on applying the same approach for *on-the-fly* detection of workloads have yielded very promising results. To deliver similar levels of accuracy, however, *on-the-fly* classification still remains subject to further investigations as well as the empirical analysis what a energy-aware scheduler can save with this classification and how it compares to other classification approaches in terms of their initially required energy budget. Further improving the discriminability of executed workloads, we wish to expand the evaluation of our approach to include a wider range of workloads as well as to incorporate the use of accelerator-based implementations that leverage the heterogeneity of the used S824L test machine.

References

[1] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka, "Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems," in *SC'13*. IEEE, 2013, pp. 1–11.

[2] B. Herzog, T. Hönig, W. Schröder-Preikschat, M. Plauth, S. Köhler, and A. Polze, "Bridging the gap: Energy-efficient execution of software workloads on heterogeneous hardware components," in *Proceedings of the 2019 ACM International Conference on Future Energy Systems (e-Energy '19)*, 2019, p. 428–430.

[3] E. C. Inacio and M. A. Dantas, "A survey into performance and energy efficiency in hpc, cloud and big data environments," *Int. Journal of Networking and Virtual Organisations*, vol. 14, no. 4, pp. 299–318, 2014.

[4] M. Terai, R. Kashiwaki, and F. Shoji, "Workload classification and performance analysis using job metrics in the k computer," 2017.

[5] M. Genkin, F. Dehne, P. Navarro, and S. Zhou, "Machine-learning based spark and hadoop workload classification using container performance patterns," in *Benchmarking, Measuring, and Optimizing*, C. Zheng and J. Zhan, Eds. Cham: Springer Int. Publishing, 2019, pp. 118–130.

[6] B. Copos and S. Peisert, "Catch me if you can: Using power analysis to identify HPC activity," *CoRR*, vol. abs/2005.03135, 2020. [Online]. Available: https://arxiv.org/abs/2005.03135

[7] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *preprint arXiv:1906.02243*, 2019.

[8] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 1916–1920.

[9] C. Cernazanu and M. Marcu, "Anomaly detection using power signature of consumer electrical devices," *Advances in Electrical and Computer Engineering*, vol. 15, pp. 89–94, 02 2015.

[10] J. Combs, J. Nazor, R. Thysell, F. Santiago, M. Hardwick, L. Olson, S. Rivoire, C. Hsu, and S. W. Poole, "Power signatures of high-performance computing workloads," in *2014 Energy Efficient Supercomputing Workshop*, 2014, pp. 70–78.

[11] C.-H. Hsu, J. Combs, J. Nazor, F. Santiago, R. Thysell, S. Rivoire, and S. Poole, "Application power signature analysis," *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pp. 782–789, 2014.

[12] H. Kim, J. Smith, and K. G. Shin, "Detecting energy-greedy anomalies and mobile malware variants," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '08. New York, NY, USA: ACM, 2008, p. 239–252.

[13] G. Jacoby, N. Davis, and R. Marchany, "Detecting software attacks by monitoring electric power consumption patterns," Patent US7 877 621B2, 2011.

[14] IBM. (2014) Power System S824L Technical Overview and Introduction. Acc. 2021-06-24. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp5139.pdf

[15] T. Rosedahl, M. Broyles, C. Lefurgy, B. Christensen, and W. Feng, "Power/performance controlling techniques in openpower," in *Int. Conference on High Performance Computing*. Springer, 2017, pp. 275–289.

[16] Microchip. (2018) MCP39F511N Datasheet. Acc. 2021-06-24. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/20005473B.pdf

[17] S. Köhler, B. Herzog, T. Hönig, L. Wenzel, M. Plauth, J. Nolte, A. Polze, and W. Schröder-Preikschat, "Pinpoint the joules: Unifying runtime-support for energy measurements on heterogeneous systems," in *2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*. IEEE, 2020, pp. 31–40.

[18] scikit-learn, "Machine Learning in Python," https://scikit-learn.org, Accessed: 2021-04-07.

[19] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, "The landscape of parallel computing research: A view from berkeley," 2006.

[20] Jill Dunbar, "NAS Parallel Benchmarks," Acc. 2021-04-07. [Online]. Available: https://www.nas.nasa.gov/publications/npb.html