A scalable framework for online power modelling of high-performance computing nodes in production

(Article begins on next page)

24 August 2024

This is the post peer-review accepted manuscript of:

F. Pittino, F. Beneventi, A. Bartolini and L. Benini, "A Scalable Framework for Online Power Modelling of High-Performance Computing Nodes in Production," 2018 International Conference on High Performance Computing & Simulation (HPCS), Orleans, 2018, pp. 300-307. doi: 10.1109/HPCS.2018.00058

The published version is available online at:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8514363&isnumber=8514305

A scalable framework for online power modelling of high-performance computing nodes in production

Federico Pittino*, Francesco Beneventi*, Andrea Bartolini*, Luca Benini*†

*Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Italy
{federico.pittino, francesco.beneventi, a.bartolini, luca.benini}@unibo.it
†Integrated Systems Laboratory, ETH Zurich, Switzerland {lbenini}@iis.ee.ethz.ch

Abstract—Power and thermal design and management are critical components of high performance computing (HPC) systems, due to their cutting-edge position in terms of high power density and large total power consumption. Many HPC power management strategies rely on the availability of accurate compact power models, capable of predicting power consumption and tracking its sensitivity to workload parameters and operating points. In this paper we describe a methodology and a framework for training two of the best-in-class power models directly on the online in production nodes and without requiring dedicated training instances. The compact power models are obtained using an online regression-based approach which can track non-stationary workloads and hardware variability. Our experiments on a real-life HPC system demonstrate that the models achieve very high accuracy over all operating modes. We also demonstrate the scalability of our approach and the small amount of resources needed for the online modeling, for both the training and inference phases.

I. INTRODUCTION

High performance computing (HPC) systems are designed to be at the cutting edge of computing power. To achieve this goal, HPC installations are characterized by high computational power density and as consequence, by high power consumption density as well as high total power consumption. Indeed HPC systems have 2-4x higher rack power density w.r.t. server and industrial datacentre installations, with a per rack power envelope ranging between 20-100 kWatts [1]. High power density and envelope are obviously critical for HPC system management and operation.

Today the most powerful supercomputer in Top500 is Sunway TaihuLight which consumes 15.3 MW for delivering 93 Petaflops. The second one, Tianhe-2 (ex 1st) consumes 17.8 MW for "only" 33.2 Petaflops. However, the power consumption increases to 24 MW when considering also the cooling infrastructure[2]. Such an amount of cooling power serves to prevent thermal issues. In fact, the performance of the processing elements is actively controlled by the internal firmware logic, which modulates chip voltage and frequency for maximizing the clock speed while satisfying power and thermal constraints. However these mechanisms are usually reactive, threshold-based and take significant safety margins: authors in [3] show that, for hot-water liquid cooled nodes, the processors are incapable of employing thermal throttling by using DVFS states to prevent the critical thermal threshold to be reached.

To solve these issues, several works in the literature [4], [5], [6], [7], [8] propose to take advantage of proactive thermal and power management strategies. These strategies all rely on the availability of compact predictive power models, capable of predicting future power consumption and, even more importantly to build a clear understanding on the sensitivity of power consumption on workload parameters and hardware knobs that can be controlled at run time. These models allow to estimate and model the power consumption of the CPUs and COREs based on workload characteristics extracted through performance counters and micro-architectural usage. Thanks to that an optimizer can leverage these models to find the maximum clock frequency to apply based on the current usage of the micro-architecture while satisfying a global power budget or thermal constraints. A different approach to solving the same problem is followed for instance by Intel's RAPL [9] by implementing a feedback loop on a direct measurement of the power of the CPUs. However, as recent works show [10], RAPL is sub-optimal and in several cases burns excessive power when it is not needed.

Instead, the compact models can be used in combination with optimization and artificial intelligence techniques to select in a robust fashion the optimal operating points from the target power and temperature and the current conditions. However, the strategies for learning these models rely on design-time parameters that cannot cope with manufacturing variability, which makes each chip different from the others, differences in deployment conditions and ageing which can induce very significant differences in compact model parameters even for nominally identical nodes. In addition, such models have been applied only to single node systems operating in a test environment and therefore cannot cope with the large number of computing elements in an HPC system in production without causing significant calibration costs (e.g. bringing the HPC machine off-line periodically for power characterization).

In this paper we describe a methodology and a framework for training advanced power models (we considered [11], [12]) as best-in-class use cases) directly on the in production nodes and without requiring dedicated training instances. More specifically, we are targeting compact power modeling at the socket level, since in the HPC systems we considered, nodes are composed of multiple sockets (packages).

The power models are derived using linear regression. The

choice of linear power models is justified by their accuracy in modeling real machines [11], [12] and comes extremely handy for applications in power and thermal management, where we need to invert the model to use it in a feedback control loop [8].

II. RELATED WORK

The topic of predictive power modeling has been studied for many years [13]. The main motivation behind the introduction of such models relied in the increasing importance of dynamic hardware adaptations, which provide an opportunity for extracting maximum performance while remaining within temperature and power limits [14], [15], [16]. The most advanced and accurate models rely on the usage of performance counters from the architecture, which can provide a large and complete set of information about the operation, not only of the processor, but of the entire system [15], [16]. The set of performance counters supported by processor vendors has undergone a continuous enrichment, together with the introduction of derived counters which are more suitable for use in prediction algorithms [17].

The application of such models on multicore architecture on servers and in cloud platforms is more recent [11], [18], [19], [20], and it is driven mainly by the critical importance of power and thermal capping in such systems. However, most of these works do not fully address the complexity and scale of a full HPC system: they are limited to analyzing only single nodes. Moreover, in most cases, modeling construction is not based on monitoring the continuous operation of a production machine, instead focusing on the study of simple, single node benchmarks in a controlled test environment. In order to overcome these limitations, a powerful and scalable infrastructure applied to a production cluster is needed. For this purpose, the usage of Big Data techniques is becoming increasingly promising in this field, as shown in [21].

The majority of models based on performance counters rely on linear regression for the model construction. The main differences between these models lie in the choice of the counters to use. Different statistical methods have been employed, ranging from Pearson's correlation of the counters with power [11] to the monitoring of the R^2 of the fit and of the multicollinearity between counters [12]. In some works (eg. [11]) the features of the linear regression model are then derived applying a non-linear transformation to the counters, however these transformations are usually derived by visual inspection and they are highly platform-specific. For these reasons, we have decided to always use as features in the linear regression model directly the performance counters.

Our main contribution is in the development of a full end-to-end procedure for model characterization and continuous adaptation which works online, during normal HPC system operation. Furthermore, we focus on scalability, using state-of-the-art big data analytics tools and framework, which allow concurrently training and updating power models for all the nodes in the HPC machine. Finally, we demonstrate that our approach is flexible and it is not limited to a single model or

parameter set, and produce highly accurate results at the scale of all the nodes of a real-life HPC system, with no perturbation on production usage.

III. FRAMEWORK

The model-learning infrastructure described in this paper is based on a distributed and scalable monitoring framework that we presented in [21].
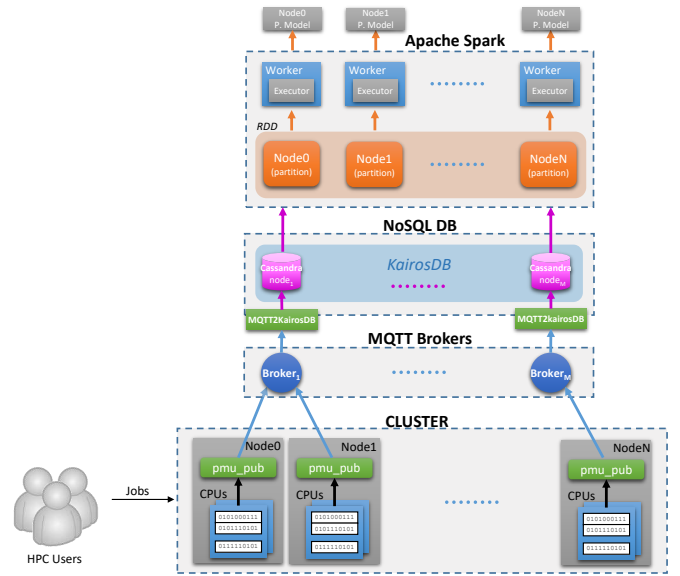

Fig. 1. Model-learning framework

With the help of the hierarchical view showed in Fig.1 we can distinguish four functional layers in the monitoring and model construction systems. These blocks include: a communication layer based on the MQTT communication protocol[22]; A set of data collection agents which run on each compute node, periodically measure physical and micro-architectural quantities and publish them through the MQTT protocol; A storage layer based on a distributed and scalable time series database (KairosDB) [23], [24] built on top of a NoSQL database (Apache Cassandra) [25], [26] which provides a mechanism to store metrics, mainly for visualization and analysis of historical data; An application layer which uses the collected data for visualization and data analytics purposes. In this paper we connected Apache Spark [27], [28] to the storage layer of the monitoring framework to build prediction models of the CPU power consumption.

For the model-learning application described in this work we employed the `pmu_pub` data collection agent [21], which runs as a service on each node of the cluster. It measures and delivers on the MQTT bus the metrics described in Tab. I and Tab. II. The collectors are configured to sample the CPU counters every 2 seconds. Using this sample rate we can achieve a measurement overhead on the target node that is less than 0.6%.

TABLE I
PER-CORE METRICS

Metric name	Description
temp	(^oC) Core temperature
instr	Instructions retired
ipc	Instructions per cycle
ips	Instructions per second
freq	(MHz) Actual core frequency
Ci	(%) time in state Ci, with i= $\{0, 1, 3, 6\}$
AVXinst	(%) Number of AVX instructions
back_end_bound	(%) slots where no μops are delivered due to a lack of required resources for accepting more μops in the back-end of the pipeline.
core_bound	(%) This metric represents how much Core non-memory issues were of a bottleneck.
L1L2_bound	(%) This metric shows how often machine was stalled without missing the L1 data cache + how often machine was stalled on L2 cache.
front_end_bound	(%) cycles where pipeline stalls caused by issues during the fetching or decoding of instructions
retiring	(%) cycles where pipeline slots are utilized by useful work
bad_speculation	(%) cycles wasted due to removing mispredicted μops from the execution pipeline
L3_bound	(%) cycles where CPU was stalled on L3 cache, or contended with a sibling Core.
issue_loss_idle	(%) number of idle slots

TABLE II
PER-CPU METRICS

Metric name	Description
Ci_pkg	(%) time in state Ci, with i= $\{0, 2, 3, 6\}$
freq_pkg	(MHz) Actual package frequency
temp_pkg	(^oC) Package temperature

A. Power prediction model

As outlined in Sec. II, the power prediction model is based on a linear regression algorithm, which is applied on the selected features. All the computation is performed by Spark on a service node (see Fig. 1), which reads data from KairosDB or from the MQTT streaming interface and it processes them using the Dataframe structure (a scalable and distributed data structure provided by the extension SparkSQL [29]).

Reading from KairosDB is accomplished by executing a query, as discussed in [21], for a specific subset of nodes and time window. Only the features relevant for the model are extracted (see Sec. IV-B), and each feature is scaled using a-priori constant values in order to lie in the range $[0, 1]$ (or to be as close as possible, since some of them are unbounded, like for example ips). The data is then stored in a Dataframe, which ensures the scalability of our approach due to its inherent scalable and distributed data structure.

We have now to distinguish two states of operation (see the pseudocode in Alg. 1):

- linear regression model training phase;
- power prediction calculation.

In the training phase, the linear regression coefficients have to be calculated. This is accomplished using the functions provided by the "ml" package from Apache Spark 2.2.0. Once the Dataframe with the model features observed for a defined time window is loaded, it is passed to the `fit` method of the

---

Algorithm 1 Training and inference phases.

```
 1: procedure TRAINING PHASE
 2:     node ← node of interest
 3:     time ← data time window
 4:     featchoice ← choice of linear regression model
        features
 5:     for all sockets on the node do
 6:         dframe ← query(node,time,featchoice)
 7:         model ← LineaRegression.fit(dframe)
 8:     end for
 9: end procedure
10: procedure INFERENCE PHASE
11:     node ← node of interest
12:     features ← linear regression model features
13:     for all sockets on the node do
14:         model ← query(node)
15:         powpred ← model.evaluate(features)
16:     end for
17: end procedure
```

`LinearRegression` class. The model coefficients are then calculated and stored in KairosDB for subsequent use in the power prediction calculation.

Once the model coefficients are trained and stored in the database, the power prediction calculation can be performed. Since we are interested in online power prediction, the model features are now read directly from the MQTT interface while the model coefficients are read from KairosDB and stored again in an object of the `LinearRegression` class. The power calculation is then performed using the `evaluate` method of the same class, and the results are broadcasted on the MQTT interface.

IV. RESULTS

A. Test bed

For our experimentation, we implemented the framework from Fig. 1 on a cluster composed by 516 nodes of a working production system (Galileo at Cineca) as a case study. Each node is equipped with two 8-cores Intel Haswell CPUs (E5-2630 v3 @ 2.40GHz) and 128GB of DRAM. The power prediction algorithms are instead run on a separate service node (Intel Haswell E5-2670 v3 @ 2.30GHz, 24 cores and 128GB DRAM), where the "Spark cluster" environment is installed.

We want again to stress out the fact that, unlike most of the previous literature on power models (for example, [14], [19], [11], [12], [20]), in this work we trained and applied our power prediction model in a production environment, where each node has a different workload which can drastically change over time, and not on benchmarks on single nodes. Fig. 2 reports an example of the sockets from two nodes in the cluster over a period of 8 days. This example clearly demonstrates that our model needs to be able to work in a wide variety of situations, both on nodes in idle state and on nodes with a high and frequently variable workload.
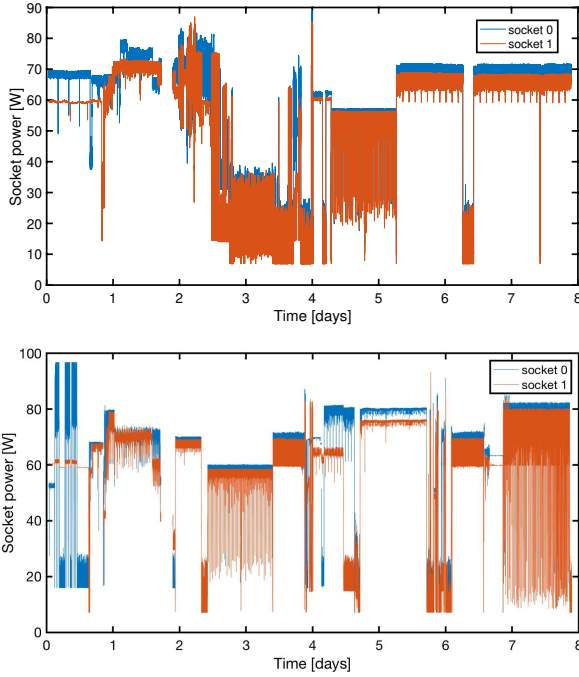
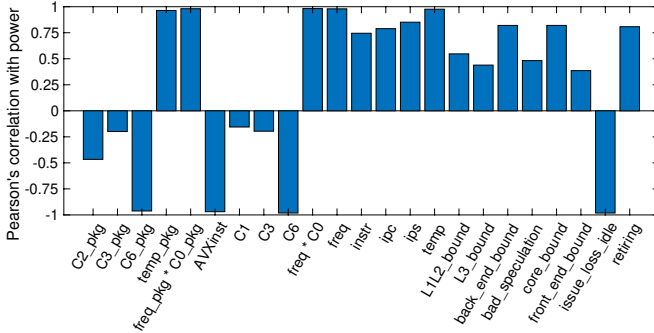Fig. 2. Trace of measured socket power for two nodes of the cluster.

Fig. 3. Pearson's correlation between the measured socket power and the most relevant available counters.

B. Choice of counters

In order to build the features for our models, we used the algorithms from [11] and [12] by employing the available performance counters provided by the CPU architecture and the counters that can be derived from them. We have performed the counters choice on a small dataset consisting of 20 hours of operation of three nodes of the cluster. We could then verify that the choice of counters does not depend on the particular node considered.

In the following we then describe how we derived the two sets of features that correspond to the models [11] and [12]. Once the two sets of features are derived, the linear regression algorithm is agnostic about how the features were selected.

As discussed in [11], the first step is to calculate Pearson's correlation of each counter with respect to the measured package power. The results are reported in Fig. 3 for all the
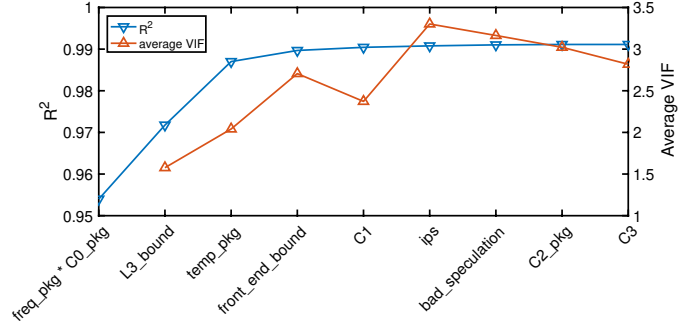

Fig. 4. Progression of R^2 (blue) and average VIF (red) when using the modified algorithm from [12].

Feature number	Model [11]	Model [12]
1	freq * C0	C1
2	temp	C3
3	C6	temp_pkg
4	temp_pkg	ips
5	freq_pkg * C0_pkg	freq_pkg * C0_pkg
6	C6_pkg	C2_pkg
7	back_end_bound	L3_bound
8	core_bound	front_end_bound
9	issue_loss_idle	bad_speculation

counters we considered (see Tabs. I-II for a description of the counters). As discussed in Sec. IV-A, in our configuration each node of the cluster is composed by multiple packages, and each of them features multiple cores (8 in our case). Since we are interested on power prediction at the package level, we have decided to average all per-core counters on a per-package basis. This allows us to greatly reduce the number of coefficients in our models, while retaining a very good accuracy (see Sec. IV-D). As is very well-known in the field of machine learning (see [30]), this averaging corresponds to a soft regularization, which helps in stabilizing the model coefficients avoiding overfitting and leads to better generalization of the model.

In order to keep the power model compact, we have chosen to use only a maximum of 9 features, which corresponds to the number of features with absolute correlation greater than 0.5 in Fig. 3, making sure not to include redundant information (as would be the case if we included for example both freq and freq * C0). The chosen features are reported in Tab. III, where their ordering is arbitrary.

In order to derive the second set of features, we have followed [12]. The algorithm works by iteratively adding features to the model until the adjusted R^2 continues to increase significantly. Once the features are chosen, it is calculated the variance inflation (VIF) of each of them with respect to all the others as a measurement of multicollinearity. If any features are identified as multicollinear using their VIF, one of the two is discarded.

Similarly to what suggested in [12], we have started the algorithm including always the feature freq_pkg * C0_pkg.

We have verified that, if we started from freq * C0 instead, the subsequent selected features remain the same, showing that freq_pkg * C0_pkg and freq * C0 convey a very similar information (as expected, since freq $\leq$ freq_pkg). The original approach by [12] was however not fully general, since the authors ended up having only two features with high VIF, and therefore they could easily exclude one of them. This is only a very special case, and indeed it was not what we obtained in our case. For this reason, we slightly modified the model by [12] by including a control on the VIF at each step of the feature selection loop based on R^2. Specifically, we had to set a maximum value for the increment in average VIF between all features from one step to another. This was necessary because otherwise all selected features end up having a very high VIF, with no clear indication of which feature is dependent on which other.

The results of the application of the modified algorithm from [12] to our dataset are shown in Fig. 4, where both the model R^2 and the average VIF for all features at each iteration are shown. In the x-axis the chosen feature at each iteration is displayed. To be consistent with the previous model, we have decided to choose at most 9 features. Finally, Tab. III shows a summary of the chosen features for the two algorithms, in an arbitrary order.

C. Models tuning on a portion of the machine

In order to explore the behaviour of the models and to tune their hyperparameters (like regularization parameter and training time interval [30]), we decided to start with a series of comprehensive tests on a reduced portion of the machine, which helps keeping excessive variability and computational time under control. For this reason, we selected only 8 nodes monitored for a total of 8 consecutive days. All results in this section refer to this dataset.
The linear regression model is defined to have no constant term, since we want our model to be able to use the provided features to predict also the idle power consumption. The regularization parameter is set to 0.001, which has proven to be a robust value to maximize accuracy.

The metric which is most commonly reported in literature is the relative error on the predicted power, and in particular the average relative error. Reporting only the average does however exclude a large piece of valuable information, since the average can be greatly influenced by few large outliers and we have no indication about how often the error lies below or above the average. For these reasons, we have chosen to report the entire Empirical Cumulative Distribution (ECDF) of the error.
Fig. 5 reports, in particular, the ECDF of the relative error obtained with the two algorithms by training a different model for each node and socket and varying the length of the training time window. As we can see, model [11] performs generally better, its accuracy improves with training time and has a lower number of outliers. In contrast, model [12] is usually worse and it does not improve substantially with the training time.

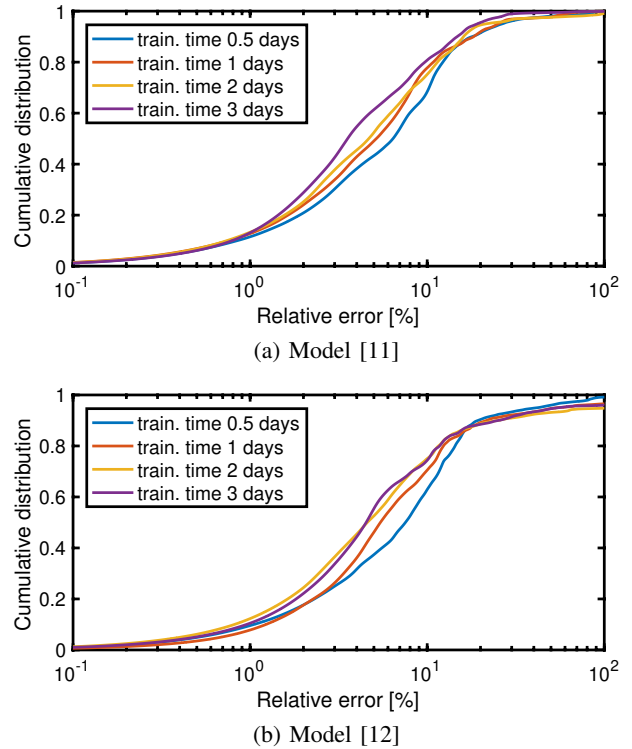

(a) Model [11]

(b) Model [12]

Fig. 5. Relative error in the predicted socket power varying the training time and the model, using the 8-nodes dataset.

The relative error is however not necessarily the most important metric to assess the model performance. Since our primary focus is on thermal management, it is instead more insightful to convert the error in predicted power to a temperature error on the package. For this purpose, we have employed previously reported measurements to directly convert a power error in temperature variation. We extracted from the target system the thermal resistance ($0.31^o K/W$) and we computed the watt necessary to increase of 1°C (3.23 W) and 3°C (16.1 W).

Fig. 6 summarizes the results of this study by employing two thresholds, one at 1°C (3.23 W) and the other at 3°C (16.1 W). Each point on the x-axis is a different training time, and for each training time and threshold they are shown the median relative number of points below the threshold together with the 25% and 75% percentiles (the error bars). We again note that the error for model [11] when training each node separately decreases significantly with the training time. On the other hand, using a single model jointly trained on all nodes provides generally a better performance when the training time interval is short, however the accuracy does not increase as significantly with the training time.
For model [12], instead, both training strategies are much less sensitive to the training time and the accuracy is generally lower with respect to model [11]. In all cases, even in the worst case scenarios generally for 90% of the points the calculated temperature error on the package is lower than 3°C, and in the best scenarios this is true in almost 100% of the points.
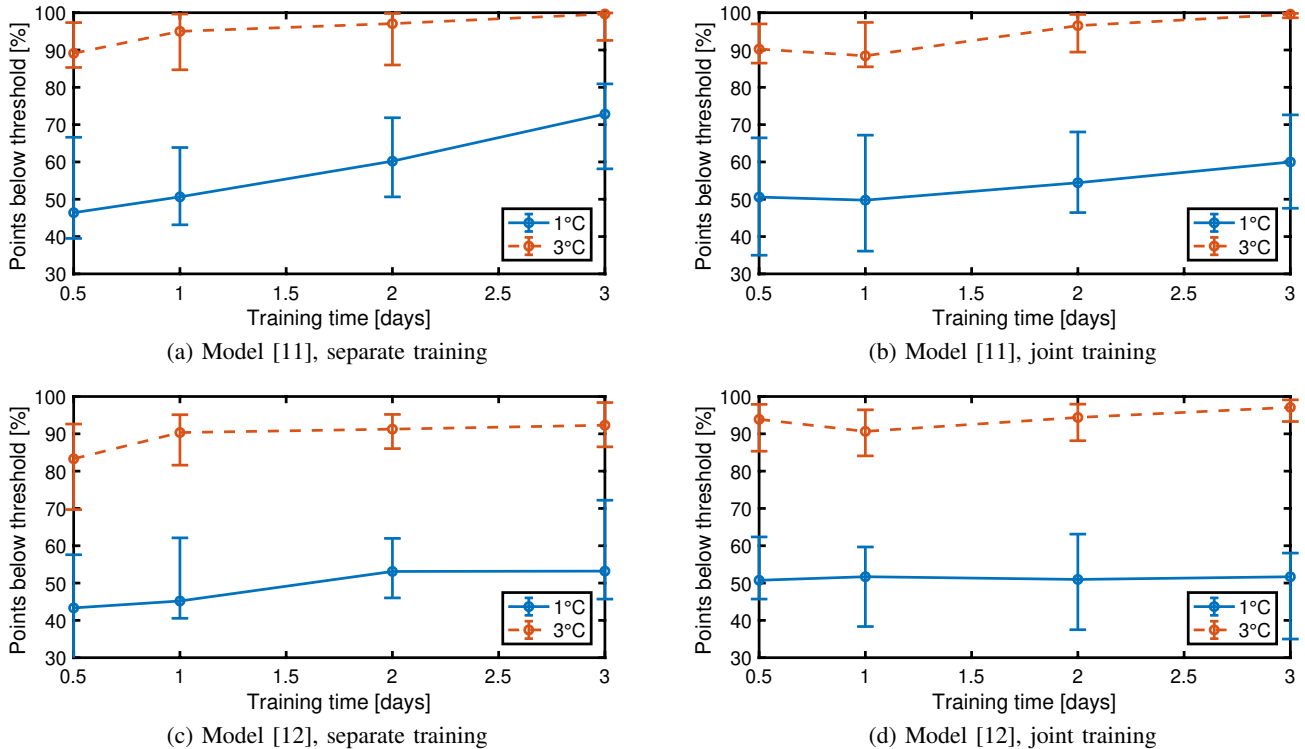
Fig. 6. Results comparing the models [11] and [12] on the 8-nodes dataset changing the training time interval, either training a different model for each node and socket or performing a joint training on data from all nodes. The two lines in each plot show, between all nodes and sockets, the median (circle) and 25th and 75th percentiles (error bar) of the percentage of points where the power error due to the estimation results in a temperature change in the package lower than a threshold (either 1°C or 3°C).

In all the results presented so far we have found that model [11] performs generally better than model [12]. However, the main motivation behind model [12] was to avoid multicollinearity between the features, in order to obtain more stable model coefficients which could then be more easily used in power optimization techniques. In order to verify this hypothesis, Fig. 7 shows the values of the coefficients for the two models when the training time is 3 days, either doing a joint training (red line) or training a different model for each node and socket (blue line). In the latter case, the circles represent the medians and the error bars are the 25% and 75% percentiles of the coefficients with respect to all nodes and sockets.

Comparing the two models, there is no evidence that suggests model [12] features coefficients are much more stable than the ones from model [11]. Coupling this observation with the fact that the relevance of the features in model [11] is more intuitive to understand and that this model usually performs better, in the following we will not consider model [12] anymore.

D. Full system results

Having selected a model and understood its limitations, we have moved to considering the application of our framework to a more challenging dataset and to evaluate its performance and scalability. For this purpose, we have selected from the cluster described in Sec. IV-A the nodes for which we had a

long record of data by using the framework described in Sec. III. The selected nodes are then 43, and correspond to a single rack. From these results we are also able to show the scalability of our framework to the entire cluster, by augmenting the data in order to reproduce the dataset that would be generated by all 516 nodes in the cluster.

For all the results in this section we have decided to use only model [11] and to train a different model for each node and socket, since we have shown that in this way we can achieve the best accuracy.

1) Model accuracy: Fig. 8 shows the model accuracy on the full cluster, as a function of the number of considered nodes, with the same approach of Fig. 6, i.e., by evaluating the temperature change in the package due to a power prediction error. Since in this dataset we have acquired data for a longer period of 20 days, we can now compare the results of longer training time intervals of 3 and 6 days.

Comparing Figs. 8 and 6, we notice that in this larger dataset the accuracy is slightly lower than in the 8-nodes dataset, given the same training time of 3 days. This is justified by the fact that we are now monitoring the nodes for a longer period of time, and therefore it is more likely for the nodes to undergo radical changes in workload. Indeed, by increasing the training time to 6 days the model accuracy is now comparable to the one obtained on the 8-nodes dataset. The observation that a longer training time is needed is further supported by our a-
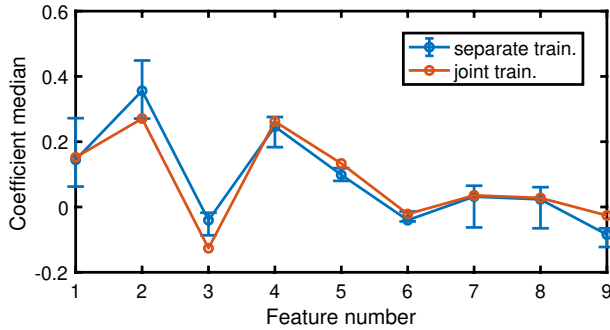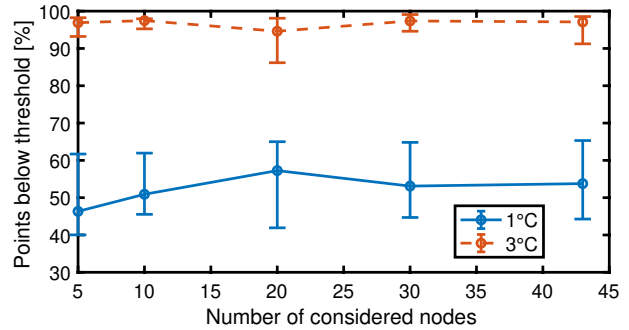
(a) Model [11]
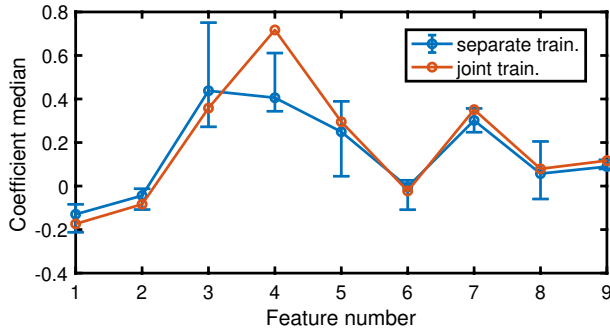
(b) Model [12]

Fig. 7. Trained model coefficients, using a training time of 3 days. In the case of separate trainings (blue lines), the circles represent the medians and the error bars are the 25th and 75th percentiles of the coefficients with respect to all nodes and sockets.

priori knowledge of the fact that the jobs running on the cluster are typically expect to last for a few days, and therefore a training interval of at least 6 days is needed to capture all variations in the nodes operation.

Moreover, as expected we note that, when considering a smaller number of nodes, the model accuracy is more variable and greatly depends on the particular nodes considered and on their operation during the current time frame.

2) Performance and resources: Having obtained an indication of the optimum model and training time interval, we have then evaluated the performance of the algorithm on our service node. We have checked that the amount of memory required is about 790 MB per node, and it scales linearly with the number of parallel executions. We have then evaluated the execution time of the training algorithm varying the level of parallelism, the results are shown in Fig. 9. Unsurprisingly, due to the limited number of cores in our service node (see Sec. IV-A), there is no increase in performance when using more than 12 parallel jobs. Using then the best result of about 6 seconds per node, we conclude that the training process of a complete cluster of 516 nodes (as in our test case) would take on average 52 minutes using 9 GB of the RAM. Note also that this computation is performed on resources that are external to the calculation resources of the cluster, and its computational demand is very limited compared to the one of the cluster. Moreover, we have shown that the training computation needs to be performed only rarely if we have a good amount of

(a) Training time interval 3 days
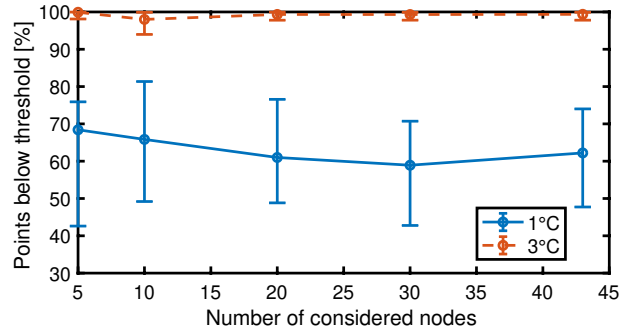
(b) Training time interval 6 days

Fig. 8. Results obtained with the model [11] on the large dataset as a function of the number of considered nodes, changing the training time interval and training a different model for each node and socket. The two lines in each plot show, between all nodes and sockets, the median (circle) and 25th and 75th percentiles (error bar) of the percentage of points where the power error due to the estimation results in a temperature change in the package lower than a threshold (either 1°C or 3°C).
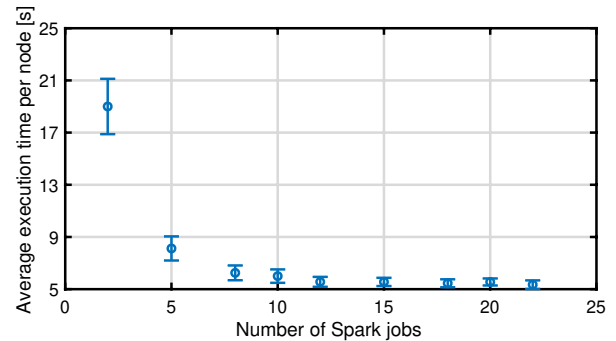

Fig. 9. Time required per node to perform the training as a function of the number of parallel Spark jobs executed on our test machine. Each Spark job could use at most 8 cores.

available data.

As far as the model inference at runtime is concerned, this is only a minimal overhead since it involves only 9 products and 8 sums per each node, socket and time sample and therefore can be done easily online.

V. CONCLUSIONS

In this work we have shown a scalable framework for power prediction in HPC clusters, discussing the importance of an accurate and efficient compact model in targeting real time power and thermal management. We have evaluated

some of the most popular power models from the literature, applied them to a real workload in a production environment and studied their accuracy in detail. We have proposed a novel method for evaluating their performance, compared the models and pointed out which one is more promising for our application.

The chosen model has a very high accuracy when trained on a relevant amount of data which cover all possible states of operation. In particular, we have obtained a median error of 5% or 2.5W on the predicted package power, and an error of 18% or 8.5W on the 95% percentile. Such power errors in our case translate to a median temperature error on the package of $0.7°C$ and of $2.6°C$ on the 95% percentile.

We have also demonstrated the scalability of our approach and the small amount of resources needed for the model, for both the training and inference phases. If the training phase is done correctly, it is not necessary to retrain the model more often than every 2 weeks, and probably well beyond this.

In addition, our framework offers a very powerful and flexible way of automatically dealing with outliers due to a model not up-to-date by continuous monitoring of the error in predicted power. Once this error for a given node and package exceeds a user-defined threshold for a certain time, a model retraining phase can be automatically triggered.

Being a scalable and very efficient framework, it is suitable for application in every HPC cluster up to date, even featuring thousands of computational nodes.

VI. ACKNOWLEDGMENTS

REFERENCES

[1] L. Gilly, "Data centre design standards and best practices for public research high performance computing centres," Ph.D. dissertation, uzh, 2016.

[2] J. Dongarra, "Visit to the national university for defense technology changsha, china," *Oak Ridge National Laboratory, Tech. Rep., June*, 2013.

[3] A. Moskovsky, E. Druzhinin, A. Shmelev, V. Mironov *et al.*, "Server level liquid cooling: Do higher system temperatures improve energy efficiency?" *Supercomput. Front. Innov.: Int. J.*, vol. 3, no. 1, pp. 67–74, Jan. 2016.

[4] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 175–184.

[5] I. Rodero, H. Viswanathan, E. K. Lee, M. Gamell *et al.*, "Energy-efficient thermal-aware autonomic management of virtualized hpc cloud infrastructure," *Journal of Grid Computing*, vol. 10, no. 3, pp. 447–473, Sep 2012.

[6] F. Zanini, D. Atienza, C. N. Jones, L. Benini *et al.*, "Online thermal control methods for multiprocessor systems," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, p. 6, 2013.

[7] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Cooling-aware node-level task allocation for next-generation green hpc systems," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 690–696.

[8] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 170–183, 2013.

[9] *Intel 64 and IA-32 Architectures Software Developers Manual*, December 2017.

[10] D. Cesarini, A. Bartolini, and L. Benini, "Benefits in relaxing the power capping constraint," in *Proceedings of the 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy Efficient HPC Systems*, ser. ANDARE '17. New York, NY, USA: ACM, 2017, pp. 3:1–3:6.

[11] M. Witkowski, A. Oleksiak, T. Piontek, and J. Wglarz, "Practical power consumption estimation for real life hpc applications," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 208 – 217, 2013, including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[12] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das *et al.*, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, Jan 2017.

[13] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, June 2000.

[14] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *SIGARCH Comput. Archit. News*, vol. 34, no. 5, pp. 185–194, Oct. 2006.

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

[16] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 46–55, Jul. 2009.

[17] A. Yasin, "A top-down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2014, pp. 35–44.

[18] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang *et al.*, "Power containers: An os facility for fine-grained power and energy management on multicore servers," *SIGARCH Comput. Archit. News*, vol. 41, no. 1, pp. 65–76, Mar. 2013.

[19] G. T. Chetsa, L. Lefvre, J. Pierson, P. Stolf *et al.*, "Exploiting performance counters to predict and improve energy performance of hpc systems," *Future Generation Computer Systems*, vol. 36, no. Supplement C, pp. 287 – 298, 2014, special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.

[20] M. Chadha, T. Ilsche, M. Bielert, and W. E. Nagel, "A statistical approach to power estimation for x86 processors," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017, pp. 1012–1019.

[21] F. Beneventi, A. Bartolini, C. Cavazzoni, and L. Benini, "Continuous learning of hpc infrastructure models using big data analytics and in-memory processing tools," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 1038–1043.

[22] OASIS, "MQTT 3.1.1 specification," dec 2010, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[23] B. Hawkins, "Kairosdb, fast time series database on cassandra," jan 2017, http://kairosdb.github.io.

[24] A. Bader, O. Kopp, and M. Falkenthal, "Survey and comparison of open source time series databases." in *BTW (Workshops)*, 2017, pp. 249–268.

[25] A. Lakshman and P. Malik, "Apache Cassandra," sep 2016, http://cassandra.apache.org.

[26] ——, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[27] M. Zaharia, R. S. Xin, P. Wendell, T. Das *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016.

[28] X. Meng, J. Bradley, B. Yavuz, E. Sparks *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[29] M. Armbrust, R. S. Xin, C. Lian, Y. Huai *et al.*, "Spark SQL: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1383–1394.

[30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.