Predicting Job Power Consumption Based on RJMS Submission Data in HPC Systems

Théo Saillant[1,2(✉)], Jean-Christophe Weill[1], and Mathilde Mougeot[2,3]

[1] CEA, DAM, DIF, 91297 Arpajon, France
[2] Université Paris-Saclay, ENS Paris-Saclay, CNRS, Centre Borelli, 94235 Cachan, France
theo.saillant@ens-paris-saclay.fr
[3] ENSIIE, 91000 Evry, France

Abstract. Power-aware scheduling is a promising solution to the resource usage monitoring of High-Performance Computing facility electrical power consumption. This kind of solution needs a reliable estimation of job power consumption to feed the Resources and Jobs Management System at submission time. Available data for inference is restricted in practice because unavailable or even untrustworthy. We propose in this work an instance-based model using only the submission logs and user provided job data. GID and the number of tasks per node appears to be good features for prediction of a job's average power consumption. Moreover, we extant this model to production context with online computation to make a practical global power prediction from job submission data using instances re-weighting. The performance of the online model are excellent on COBALT's data. With any doubt this model will be a good candidate for the achievement of consistent power-aware scheduling for other computing centers with similar informative inputs.

Keywords: RJMS · Job scheduling · Power consumption · Machine learning

1 Introduction

Power efficiency is a critical issue for the goal of ExaFLOP performance. It will be impossible to run applications at such a scale without a dedicated power plant if High-Performance Computing (HPC) systems are not more power-efficient [3]. Minimizing electricity consumption to reduce production costs and environmental issues is of ever increasing importance. Created in 2007, the Green500 ranks the 500 most energy-efficient computer systems to raise awareness other performance metrics.

P. Sadayappan et al. (Eds.): ISC High Performance 2020, LNCS 12151, pp. 63–82, 2020.
https://doi.org/10.1007/978-3-030-50743-5_4

1.1 Constraints for Job Scheduling

One potential direction to improve the power efficiency is to better understand user behaviors and to create incentives to reward the use of power-efficient applications using software. The Resource and Job Management System (RJMS) can be used in this case. If the RJMS is aware of the power consumption, an energy-budget scheduling policy [5] to avoid power peaks or to address the intermittent nature of renewable energies can be designed.

Nevertheless, power consumption is rarely known in advance, even by the user, so an alternative solution is to add power consumption forecasting facilities to the RJMS. Since the only job data available at the stage of submission is what the users provide to the RJMS on the resources their jobs need for allocation and running, one solution may be to use the RJMS submission data to predict the power consumption of the submitted jobs before scheduling, as already proposed by [2]. Many studies have been carried out on job power consumption estimations [17,18], however these have often made use of application type data. Application type data is not generally available due to confidentiality issues and can also be untrustworthy, because users may falsify data (e.g. by renaming the executable file) to take advantage of the scheduling policy, for power-aware scheduling.

A particularly useful functionality would be to use only the RJMS submissions data to predict the power consumption of a job. The power consumption of the whole computer center can then easily be monitored.

1.2 Related Work

The use of RJMS data in this type of problem has already been investigated; using application types [3] or symbol information [20] to make predictions. In these works, the data is not restricted to submission data.

Online model to forecast the elapsed time of the job using only the data given at submission and the current user's usage is proposed by [7] so that the RJMS can use backfilling more efficiently [13]. This estimation is designed for backfilling and may not be good for power-aware scheduling. An estimate for memory usage and run time using only submission data has been proposed [19]. Although these papers did not estimate power consumption, the used inputs suggest that user information is needed to provide a practical estimate when application types are not available.

Except in a few cases, the instantaneous power consumption of a user and the whole computer center can be predicted with workload information as the number of nodes, components used by the user's jobs and runtime [16]. In our case, the time evolution of power consumption within a job is not available in the log files. In a further study [17], submission data to predict job duration, and not power consumption, are used with the executable name as input data.

1.3 Contributions

The main contributions presented in this paper are first an instance-based model to predict average power consumption of a job per node using only the data

submitted to the SLURM RJMS. An online model is then proposed, easy to use and to maintain in production while a weighted model is introduced to predict the global power consumption of all jobs.

Moreover this work shows how it is possible to build an efficient model to forecast the power consumption based on the exploitation of a historical log database from the SLURM RJMS data collected from the industrial computer center COBALT that is only composed of user inputs of jobs and associated energy consumption measures.

Submitted data appears to be sufficient to provide a good estimate of job power consumption for the RJMS. This can be used in power-aware scheduling with our generic model because job submission is redundant. This model may be used in other industrial HPC facilities for power-aware scheduling because it uses only the data that is necessary for scheduling and because it is easy to maintain, but further tests must be made using data from other computing centers.

The paper is organized as follows: we first extract and pre-process log data from SLURM RJMS [21] and we introduce an instance-based model to process the submitted data as categorical inputs. The model is then adapted to remove biases and to handle streamed data. The final section presents the results.

2 Extracted Data and Preprocessing

2.1 The COBALT Supercomputer and The SLURM RJMS

The data used for this application are collected from the COBALT[1] supercomputer, more precisely, from its main partition which is composed of 1422 nodes ATOS-BULL with Intel Xeon E5-2680V4 2.4 GHZ processors that have 2 CPUs per node and 14 cores per CPU. The Thermal Design Power of each CPU is equal to 120 W.

The energy accounting and control mechanisms are implemented within the open-source SLURM [21] Resource and Job Management System (RJMS) [8]. The data are recorded from accounting per node based on the IMPI measuring interfaces [8]. IMPI collects data on the consumed power from all the components (e.g. CPU, memory, I/O, ...) of the node, temporally aggregates it and returns the consumed energy during an elapsed time to SLURM. As it is impossible to differentiate between jobs running on the same node, so it was decided to exclude jobs that did not have exclusivity on a node.

The collected dataset is the resulting logs of SLURM submission data of 12476 jobs run on the supercomputer over 3 months at the beginning of 2017 and their respective consumed energy. The jobs that do not have exclusive usage of a node or for which the consumed energy is null are filtered out.

2.2 From Raw Data to Relevant Features

There are two potential outputs to predict: the elapsed time and the total consumed energy which are both available once the job is finished. For various

[1] https://www.top500.org/system/178806.

reasons (e.g. failure of jobs or sensors), null value can be sometimes returned for energy consumption. Only non-zero values of energy consumption are here considered.

Three groups of information provided to SLURM may be used to predict the output (a summary is provided in Table 1)

1. **Information on the user:**

   User Identifier (or UID) is a number identifying the user that submits the job. 200 separate UIDs were observed over the 3 month period.

   Group Identifier (or GID) characterizes the users that belong to the same company or community sharing the same group. This number allows the inclusion of an *a priori* on what type of job the user runs. 30 unique GIDs were observed over the selected period.
2. **Type of resources required by the job:**

   Quality of Service (QoS) sets the maximum timelimit, and discriminates between test and production jobs.

   Timelimit can be set by the user to benefit from backfilling. This is a continuous system variable, but only 520 distinct values were used over the 3 month period (430 by the same user), showing that users often reuse the same value. Hence, we chose to discretize this variable by taking only the number of hours that are needed (c.f. Table 1).
3. **Computing power quantities required by a job:**

   Number of tasks in parallel is defined by SLURM with option -n (e.g. the number of MPI processes if MPI is used)

   Number of cores per task is defined by SLURM with option -c and is used for threading models or if an MPI job needs more memory than is available per core. This information is combined with the number of tasks to form the number of nodes required and is not stored.

   Number of nodes: SLURM combines the number of tasks and the number of cores per task to define the number of nodes needed but the user may specifying this directly.

SLURM logs may also be useful for prediction:

Date of submission of the job. This cannot be used directly as input since no future job will have the same date. However, some features can be computed based on the time of day the submission was made (c.f. Table 1).

Final number of nodes that the SLURM allocated for the job. This is the same as the number of nodes required in our data.

Start date of the job can differ from the submission date if the job has to wait to be run, but it is set by SLURM and not the user, so it is not used. The same holds for the **end date**.

Executable name could be used in some cases to identify the type of application the job is running. However, it can be irrelevant ('python' or 'a.out' are extreme examples) and users may take advantage to manipulate SLURM if it is used to define scheduling policy. Hence, it was decided to ignore this in our model.

Table 1 summarizes the model's inputs and the potential outputs of interest. Although the number of cores per task is unavailable as it is not memorized by SLURM, it is an interesting value. The average number of tasks per node (tasks/node in Table 1) can be computed as an equivalent quantity. Redundant features, such as the required number of nodes that is given by the SLURM RJMS but is almost always equal to the final number of nodes, are removed.

Table 1. Synthesis of the relevant handcrafted input features and outputs of SLURM for the studied model.

Feature	Meaning	Comment
	Potential raw inputs	Information before allocation
UID	User IDentifier	Anonymized and unique identifier
GID	Group IDentifier	Project membership identifier
QoS	Quality of Service	Indicates if job is in test/production
#nodes	Number of nodes allocated	Redundant with requested number
#tasks	Number of tasks in parallel	E.g. number of MPI processes
submit	Date of submission by the user	Cannot be used directly
timelim	Time limit before a job is killed	Cannot be used directly
	Computed features	Knowledge incorporation
tasks/node	Number of tasks per node	Manually created features
submit_h	Hour of submission in the day	Relevant information from submit
timelim_h	Limit duration in hours	Relevant information from timelim
	Outputs	Given after the job is finished
elapsed	Time elapsed	True duration of the job (wall time)
energy	Total consumed energy by the job	Aggregate temporally and by nodes
	Target	Model output
meanpow	Average power consumption per node	Computed as Eq. (1)

2.3 Target and Problem Formalization

The elapsed time and the power consumption are the two unknown values needed before the job runs to improve the management of power consumption by the RJMS.

The elapsed time inference which has been a subject of interest in several papers [7,13,17] improves the backfilling of the scheduling policy so that resource usage is maximal at any time.

The energy usage value returned by SLURM is the total consumed energy used by all the nodes for the entire job duration. The energy consumption

increases by definition if the elapsed time increases or if the number of nodes that a job uses increases. The total energy grows approximately linearly with the number of nodes and elapsed time. However, this assumption has some limitations, as it implicitly means that the power consumption remains constant when the job is running and each node uses the same amount of resources over time. Although this assumption is strong, it is not far from reality for the majority of jobs, as shown by [1], and it can be removed only with time-evolving data inside jobs that is not available.

If the number of nodes and elapsed time are not provided, the meaningful consumption statistic able to be predicted given the information collected by SLURM is the average power per node. The average power per node, denoted by meanpow, is defined and computed as:

$$\text{meanpow} = \frac{\text{energy}}{\text{elapsed} \times \#\text{nodes}} \tag{1}$$

Once a model returns the average power per node, the job's power consumption can be computed by multiplying it by the number of nodes and used in a monitoring policy as the estimation $\tilde{P}_{comp}$ for budget control [5] or powercapping. If the elapsed time is given (by other models like [7]), the consumed energy can also be predicted under the linearity assumption.

Most of the previously proposed methods use standard machine learning models from the SciKit-Learn python library [14], such as decision tree, random forest for [3] or SVR for [16]. Those models provide interesting results, but all of these rely on several parameters known to be difficult to tune and they assume regularity in the input space that may not exist in our case. Our first motivation and our contribution are to propose an alternative model that requires fewer assumptions and that works at the same time efficiently.

3 Instance Based Regression Model

3.1 Inputs as Categorical Data

Using all the available data, Fig. 1 shows four empirical distributions of the approximated average power per node for jobs with the same number of cores per task. We observe that the distributions are well separated with respect to the power. It shows that the number of cores per task is already an efficient criterion to estimate average power per node for certain jobs. This is particularly the case for the most power consuming jobs (reaching 300 W/nodes), which most likely use one core for each task, and those using 7 cores per task, which mainly use half of the full power. This is in fact expected as the number of cores assigned to the same tasks generally depends on the threading model of the application, which implies a different power consumption. The Gaussian like distributions contain interesting and useful information. Hence we infer that a low complexity model may be useful for modeling. Combined with other inputs, such as UIDs, we expect a good discrimination of power usage for any submitted job can be made based on a few internal parameters.

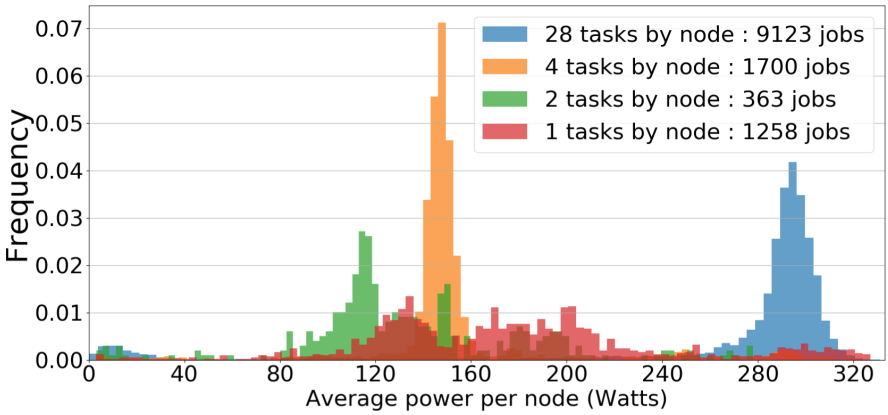

Fig. 1. Distributions of average power per node. Each histogram is computed for jobs using the same number of tasks by node. Full MPI jobs use 28 tasks by node.

In our application, the input features are either categorical or numerical, for example:

- The metadata related to the chosen QoS, the user and group identifiers (UID, GID) are categorical and thus their values (numerical or not) cannot be ordered.
- Other features are numerical and describe two types of information; discrete (the number of nodes or tasks) or continuous related to date or time (submit, start, end date of the job, duration and timelimit).

However, the discrete numerical features (number of nodes, tasks, or their ratio) may also be considered as categorical variables. For example, an application's performance depends on the number of cores and is sometimes optimal when the number of cores verifies arithmetic properties, e.g. LULESH 2.0 should be used with a number of MPI processes that is a perfect cube [11]. An application running with $27 = 3^3$ cores is likely to be different to a plausible OpenMP application using 28 cores (28 is the number of cores on a COBALT's node). Full MPI jobs use one task for each core while full OpenMP jobs use one task for the whole node. It shows that the threading model imposes the number of tasks per node.

It then seems more relevant to consider the number of nodes or tasks as a class of job or a category. Only 55 unique values were observed for requested nodes in the data when the range of possibilities is theoretically $\simeq$1000, which confirms the discrete and categorical behavior of the number of nodes or tasks. Although time related data is continuous by nature, we choose to discretize it at an hour level to have categories.

In the end, we transform all available inputs as categorical. We then propose a data-model to predict the average power consumption per node (meanpow) of any job, using only categorical inputs.

3.2 An Input-Conditioning Model

Categorical data is generally hard to handle in machine learning because all possible combinations of input values must be considered for optimization and this grows exponentially with the number of inputs. However, though a large number of combinations are possible, only a few are observed in our dataset. Submissions may be redundant and this is a motivation to use an instance-based regression model.

Algorithm 1. instance-based model Training

instance-based model Training

Require: $FeatSelected, Estimator, trainset$
  $Values \leftarrow Unique(FeatSelected(trainset))$ ▷ $FeatSelected$'s values in $trainset$
  for all $val \in Values$ **do** ▷ Group by jobs' value of $FeatSelected$
    $\mathcal{J}_{val} \leftarrow \emptyset$
    for $j \in trainset$ **do** ▷ $\mathcal{J}_{val}$ = Jobs where $FeatSelected$ match val
      if $FeatSelected(\{j\}) = val$ **then**
        $\mathcal{J}_{val} = \mathcal{J}_{val} \cup \{j\}$
      end if
    end for
    $OutputDict(val) \leftarrow Estimator(\mathcal{J}_{val})$ ▷ Estimate output value for val input
  end for
  return $OutputDict$

Algorithm 2. instance-based model Prediction

instance-based model Prediction

Require: $FeatSelected, OutputDict, job$
  return $OutputDict(FeatSelected(\{job\}))$ ▷ $OutputDict$ for $FeatSelected$ of job

An instance-based model computes a prediction by searching comparable instances in a historical training set. The simplest case of instance-based learning is Rote-Learning [15] as the nearest neighbor approach with a trivial distance [4]. The prediction for a given job is an estimator computed on the subset of the training instances that share some inputs as already proposed in [17].

Let $\mathcal{J}$ denotes our job training dataset. Each job $j \in \mathcal{J}$ is a combination of observed values for the features described in Table 1. Rote-Learning is a supervised problem for data as $(X_j, Y_j)_{j \in \mathcal{J}}$. X_j is the vector containing selected input features (i.e. a subset of the submission data as seen by SLURM) of job j used to predict Y_j. X_j is referred as the "job profile". Y_j is the target output of job j computed with any available features. In our case, it is the average power per node called meanpow as defined by (1). This is a regression task of Y_j given X_j since Y_j is real valued.

Common regression models make assumptions regarding the behavior of Y given X through a linear hypothesis or a kernel method like SVR in SciKit-Learn [14] and assume implicitly that X is either a continuous or a binary variable. In

our case X is discrete and these models can be used consistently with "dummy indicators" for each possible modality of a categorical variable. However, the input space dimension grows at the rate of the number of unique values for categories, which makes these models impractical.

On the contrary, the Rote-Learning regression model computes an estimator of the target for the jobs in the training set that have the same job profile as described in the pseudo-code 1 for training and 2 for prediction. We introduce two functions to tune how the predictions are computed. $FeatSelected()$ is a function that extracts job profiles $(X_j)_{j\in\mathcal{J}}$ that are the values from a fixed subset of inputs from job set $\mathcal{J}$. $Estimator()$ is a function that takes a list of jobs with the same profile X_j and computes a chosen estimator as prediction. After training, we return $OutputDict()$ as a mapping or dictionary that returns the prediction of any job j having the profile X_j extracted with $FeatSelected()$. When the job has a profile X_j that is not found in the training set, a prediction for a subset of the profile X_j can be made by another Rote-Learner to handle this case or a default value can be returned.

It is well-known that the Rote-Learner is the best unbiased estimator as discussed by [12]. This means that if no prior knowledge is incorporated into another model, the Rote-learner has a smaller loss. Despite this strength, the Rote-learner is rarely used in machine learning because of memory issues and for statistical reasons: the number of samples for each combination of inputs must be sufficiently large and this is rarely the case.

In our case, the number of unique observed inputs is limited in our dataset because the number of samples for a combination of inputs is large enough, hence there are no memory issues. Training time is then short because it is basically the time taken to compute the $Estimator()$ multiplied by the number of unique job profiles in the training set.

In the sequel, we use the arithmetic mean of average power per node as $Estimator()$. This minimizes the Root Mean Square Error (RMSE) for average power per node, which is then the loss function used to evaluate the possible models. In our framework, $Estimator()$ is defined as:

$$OutputDict(val) = Estimator(\mathcal{J}_{val}) = \frac{1}{\operatorname{card}\mathcal{J}_{val}} \sum_{j\in\mathcal{J}_{val}} \frac{\text{energy}_j}{\text{elapsed}_j \times \#\text{nodes}_j}$$

$$\text{with } \mathcal{J}_{val} = \{j \in \mathcal{J} | FeatSelected(\{j\}) = X_j = val\} \text{ for the job profile } val.$$

3.3 Variable Selection

The number of internal parameters that the Rote-Learner has to learn during training is the number of unique job profiles in the training dataset. For a fixed training dataset, this number depends of the choice of subset of inputs that defines the job profile. If this number increases, the model complexity also increases. The model complexity is a statistical concept that quantifies the ability of the model to fit complex phenomena, even in the case of simple noise. But a low complexity model is able to generalize for new data. For this reason, complexity and then the job profile $FeatSelected()$ definition must be carefully chosen.

In our application there are less than 10 features, so the number of possible input feature combinations needed to define $FeatSelected()$ is quite low, and we can exhaustively test all the features subsets one by one and retain only the best one. In this work, a cross-validation procedure is used to find the best inputs: we split our data into two parts, the training set is the first two months of data and the test set is the last month.

This procedure allows the identification of SLURM information pieces which are meaningful. However the computations can be time consuming. Once we empirically find the best inputs, we use only these for the following models as job profiles without repeating the process of finding the most relevant inputs. We discuss the performance results in the experimental part of Sect. 5.

At its best, the resulting model predicts the average power consumption per node of any job. Nevertheless, this objective is not monitoring the global power consumption. Certain jobs matter more than others and they are presented one by one in practice with no training time, which motivates the improvements of the following section.

4 Global Consumption Practical Estimation

4.1 Weighted Estimator for Global Power Estimation

In practice, jobs that run for the longest on many nodes contribute the most to the global power consumption of a computer center. Moreover, we observe a correlation between the duration of the jobs and the average power per node. The scatter plot in Fig. 2 shows that jobs running for less than one minute consumed less power than the others. A possible reason is that the jobs are first setting up parallelism and reading data from disks. This phase is not generally compressible and does not consume significant amounts of power. If a job is short (test, debug job or crashed), perhaps less than one minute, this phase becomes non-negligible and may then lower the average power consumption, which explains the observed bias.

However, each job has the same contribution to the mean estimate used in the previous section. Short jobs disproportionately lower the mean estimate that is defined in Sect. 3.2, despite their limited contribution to the global consumption. This is why jobs should be weighted by their total consumed node-time (number of nodes multiplied by elapsed time) when computing the mean for global consumption estimation. One method is to sum the total consumed energies then divide by the sum of their total node-time instead of dividing consumed energies individually then taking the mean. More formally, $Estimator()$ must be chosen as:

$$Estimator(\mathcal{J}) = \frac{\sum_{j \in \mathcal{J}} \text{energy}_j}{\sum_{j \in \mathcal{J}} \text{elapsed}_j \times \#\text{nodes}_j} \tag{2}$$

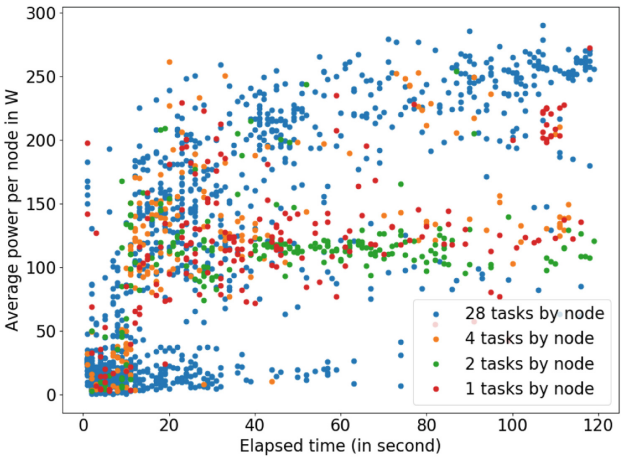

Fig. 2. Scatter plot of jobs less than two minutes long, short jobs consume less power.

4.2 Online Computations

Previous section has presented a model which computes offline the estimation of the arithmetic mean: training and prediction are two distinct and successive steps. Once training is done, the model is fixed and used to predict the power consumption of the job.

In the case of job scheduling, data is presented to SLURM as a stream of logs containing information on submitted jobs and the previous two step approach has a major flaw. A model used for prediction does not continue to learn: for a job's profile that was not present in the training data, it can only return a default value at best every time it appears. The whole model can be regularly retrained but it is then necessary to memorize all the recent data seen by SLURM in prevision of the next training round. This approach has other drawbacks: if the rounds are too frequent, a training set may be too small and if they are too rare, a lot of data has to be memorized and the prediction may be worse before the rounds.

Thankfully, the arithmetic mean used as $Estimator()$ can be straightforward to compute online and lots of approaches exist in the literature [10].

If $OutputDict(X_i)_m$ is the mean estimator at the $(m+1)^{th}$ occurrence of a job with inputs X_i and average power per node Y_i, it can be updated independently once the job is finished as $OutputDict(X_i)_{m+1} = \frac{m}{m+1} OutputDict(X_i)_m + \frac{1}{m+1} Y_i$. The counter m and current value $OutputDict(X_i)_m$ only should be maintained to compute the next value when a job ends. This is called a cumulative moving average or CUMSUM [10]. This is referred to as an online model because the model is continuously training itself, and a training round is not required.

However, this CUMSUM model gives equal weight to old and recent observations of a job profile X_i, and thus the expected job average power per node is expected to always be the same. This is not always true: a group of users may

suddenly change the applications they use which may impact the power consumption. A good way to account for such a trend-shift is to compute a moving average defined as a mean of recent data within a time-window [10]. Once again, memorization of recent data is required. However, we need to set the number of recent observations used to compute the moving average, this may not be easy.

An Exponentially Weighted Moving Average (EWMA) introduced in [9] and [10] for time series analysis is an nice way to compute a weighted moving average without memorizing any recent data. This method weights recent data more heavily than old data according to an exponential decay and then computes the mean. The exponential decay allows the moving average value to be updated with a simple formula:

$$OutputDict(X_i)_{m+1} = \alpha OutputDict(X_i)_m + (1-\alpha)Y_i \tag{3}$$

where $\alpha \in [0,1]$ is a hyperparameter to be chosen (values approaching 0 indicate lesser influence from the past). A custom weighting is required to remove underestimation of the computed estimator for global power consumption estimation.

4.3 Exponential Smoothing for Weighted and Streamed Update

As stated before, EWMA has the big advantage of memoryless updating but it must be weighted in the update formula (3) for global power consumption estimation. Previous online estimators were initially designed and used for time series analysis [9,10]. To weight them consistently, as in Sect. 4.1 and keep them online, we formally define their associated time series and modify it slightly.

At any time, the value of $OutputDict(X_i)$ is the last estimation of meanpow for a job with the profile X_i since a job with profile X_i ended. The value of the estimate $OutputDict(X_i)$ changes only when a job with profile X_i ends. As it is an evolving mean, it behaves like a trend estimation of the series of meanpow of jobs with profile X_i ordered by end date. Each job contributes in the same way to the future estimation. The contribution to the estimation of a job with profile X_i depends only on which rank it ends. The job's contribution to the online estimation does not depend on its node-time contrary to Sect. 4.1. An example of the series and its estimation by EMWA are given in Fig. 3. We observe that the EWMA is lowered by the low node-time jobs with low meanpow that have the same weight the highest node-time jobs because the series is agnostic to this quantity.

We propose a novel way to account for the needed weighting of the job without changing much our online estimation. The idea is to generalize and compute trend estimate on another series that is irregular. It is the same previous series of the average power per node of jobs with given profile X_i ordered by end date but the intervals between two successive finished jobs is the node-time of the first job, as if a job must wait the node-time of the last before starting. The resulting estimator is a continuous smoothing of this irregular time series parametrized by a node-time constant and can be used as before for online estimation but jobs with lowest node-time will not change the trend estimation as

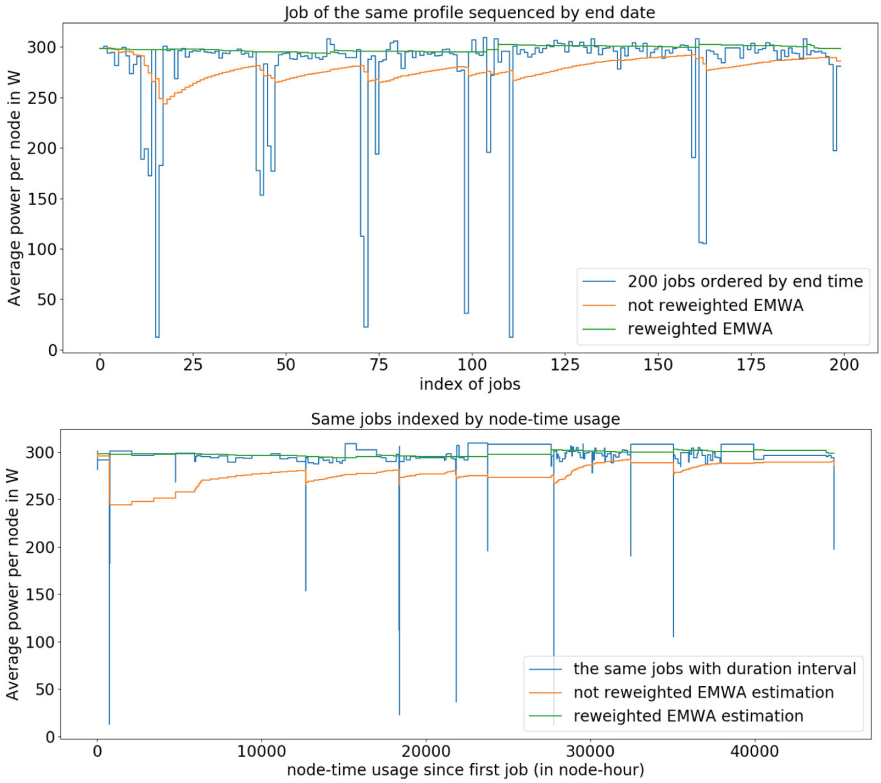

Fig. 3. Up: Series of the average power per node of 200 jobs with the same profile and its estimation by classical and reweight EWMA. **Bottom:** The associated irregular series used to weight the jobs according to node-time and the same EMWA estimations series. EWMA hyperparameters are $\alpha = \exp(\log(0.5)/20)$ (a job contribution is halved after the 20 next ended jobs) in regular case and $\tau = 4000$ node-hour in irregular case.

much as the ones with highest node-time. The adaptation of CUMSUM replaces m by the sum of the previous jobs node-time with inputs X_i, and in the moving average case the recent job are weighted by their node-time for example. The irregular series deduced from the previous example are given in Fig. 3. The short jobs have almost no influence on the current re-weighted estimate even if their meanpow value are extreme. On the contrary, it is clear that the classical EWMA strongly underestimate the irregular series meanpow because of them.

We propose to apply this adaptation to EWMA so that our estimation is memoryless and weighted correctly. We compute directly the weighted estimator without computing the irregular series by slightly modifying the previous estimator formula (3) to take into account of the node-time of the current ending job. EWMA is generalized as exponential smoothing and computed for irregular time series in [22] or [6], the update formula uses variable α to account for the irregular time interval thanks to the memoryless property of exponential:

$$\text{EMWA}(Y)_{t_n} = e^{-\Delta t_n/\tau}\text{EWMA}(Y)_{t_{n-1}} + (1 - e^{-\Delta t_n/\tau})Y_{n-1} \tag{4}$$

t_n is the time of the n^{th} sample, $\Delta t_n = t_n - t_{n-1}$ the length of the $n-1$ interval between samples, and τ a chosen time constant of exponential decay.

Applied to the trend estimation of the irregular time series, (4) formula shows that α in (3) must be replaced by $e^{-\Delta t_i/\tau}$ to weight the job i according to its node-time Δt_i. τ must be in the order of expected node-time value for several meaningful jobs. Due to (4), it is not necessary to compute and maintain the irregular time series, (4) is used when a job ends and the current estimation $OutputDict(X_i)$ is updated by computing node-time $\Delta t_i = \text{elapsed}_i \times \#\text{nodes}_i$ and setting $\alpha = e^{-\Delta t_i/\tau}$ in (3). Our method benefits from both the advantages of EWMA and the weighting correction for global power estimation. Benefits of our approach are discussed in the last experiment of the next section.

5 Numerical Results and Discussion

5.1 Offline Instance-Based Model

The proceeding described in Sect. 3.3 is run using the instance-based model offline introduced in Sect. 3.2 to determine the best job profile X_i. For each possible job profile, the model is trained using a training set containing the 8000 jobs of the first two months. Then the RMSE is computed for a testing set of 4000 future jobs from the next month.

It appears that a significant part of the jobs in the testing set show a combination of inputs that were never observed during training. In this case the model does not return an output value if the job profile is not seen previously in the training set. For a fair evaluation of any choice of job profile, we need to avoid handling the case where a pretrained model does not return an output because the profile is not known by the model. For that, we first extract a small test subset from the initial testing set composed of 1022 jobs for which their profiles are present in the training set. This is so that any model returns an output value no matter what job profile it uses.

We illustrate the bias-variance trade-off by showing the best choice of job profile that has a given number length with the lowest score (here the RMSE) and the number of unique job profile values observed in training set is shown as "diversity". Diversity is a simple way to approximate the complexity of the model to highlight bias-variance trade-off. We present the results for the small testing set in the first column of Table 2. In the special case where the job profile has zero inputs, the model always returns the mean of the average power per node of all jobs in the training set.

The best prediction requires features that identify the user, because users tend to submit the same jobs. UID is first chosen but the number of tasks per node improves power estimation and is more general when combined with GID instead of UID. Indeed, diversity is lower when the model uses GID instead of UID, and GID still indicates well enough that the jobs may be similar as

Table 2. Variable selection by cross-validation and results. The score is the RMSE (lower is better). Diversity is the number of memorized instances after training.

	Results on small test			Results on large test	
#	Best combination	Score	Diversity	Best combination	Score
0	(returns the mean)	78.04	1	(returns the mean)	89.87
1	UID	46.17	≃150	UID	44.85
2	GID, task/node	43.98	48	GID, task/node	**43.31**
3	GID, task/node, timelim_h	**43.63**	217	GID, task/node, QoS	43.83
4	Add QoS	43.78	232	Add timelim_h	44.16
5	GID, QoS, #nodes, #tasks	45.09	245	Add UID (all features)	45.49
6	Add timelim_h	45.53	475	(no more features)	–
7	Add UID (All but submit_h)	47.55	578	(no more features)	–
8	All features	52.84	1981	(no more features)	–

part of the same project given they have the same number of tasks per node. Surprisingly, adding the hour part of timelimit improves the results although it drastically increases the diversity. However, adding more inputs to the job profiles worsens the results, and the effect of over-fitting is stronger as diversity increases. In particular, the number of nodes and tasks by themselves seem to be not relevant for prediction of the power consumption as these parameters are always selected together. QoS does not seem to be informative on power usage. The hour of submission is the last selected feature showing that the type of job is the same no matter what the hour in the day is, which can be explained by auto-submissions.

The number of nodes, tasks and the submitted hour can have a large range of unique values that substantially increase the diversity which means they tend to produce over-fitting. In our experiment, this is observed when the result does not improve if these values are accounted for. But the reduced testing set is constructed only with jobs that have a combination of all these inputs values in the training set. To get more robust results about other choices of input features we reduce the space of possible job profiles which increases the number of jobs in the testing set with a profile in the training set. As these parameters seem not to be relevant for prediction, they are removed, and a larger testing set of 2216 jobs is constructed with the combination of inputs without these omitted values. The results are given in Table 2 in the second column.

The RMSEs are of the same order of magnitude as they do not depend on the size of the dataset. The same behavior in variable selection is observed, except that timelimit is no longer relevant, even selected after the QoS (and we can only choose up to 5 features as the others are removed). This difference may be explained by the strong selection of which jobs are included in the small testing set slightly favoring over-fitting.

From these observations we conclude that the GID and the ratio of number of tasks and nodes are the best choice of features to predict the average power consumption per node with any model for data on COBALT. The resulting model of this choice of features will be called IBmodel for Instance-Based model in the next sections.

5.2 Comparison with the Offline IBmodel

We compare the IBmodel with models currently in use and proposed by [3] and [19]. We focus on models based on trees, Decision Tree Regression (DTR) and Random Forest (RF), that are well-known to handle better inputs from categorical features. We also add the Gradient Boosted Regression Trees (GBRT). For these last two models, we increment the number of tree estimators and retain only the best results. We also compare the IBmodel with results from Support Vector Regression (SVR), as used by [17], choosing the best SVR parameters by manual tuning. The SciKit-Learn library [14] is used to run and train the models.

Table 3. Comparison with other models. Score is RMSE (lower is better). **Score (all):** result with all the input features for the large test set. **Score (selected):** results with input features being GID and task/node. As RF training is not deterministic, it is run 100 times, then the mean score and standard deviation are given.

SciKit models	Tested parameters	Score (all)	Score (selected)
DTR	Pure leaves, MSE criterion	48.80	43.31
RF	Pure leaves, 0 to 50 trees	45.79 (0.15)	**43.14 (0.07)**
GBRT	max-depth 5, 0 to 300 trees	44.40	**43.10**
SVR	rbf kernel, $C = 1000$, $\gamma = 0.01$	53.58	45.79
IBmodel	$X_i = (\text{GID}_i, \text{task/node}_i)$	**43.31**	43.31

At this stage, the IBmodel is not designed to return an output in case of unknown job profile. So our tests are run on the large testing set of 2216 jobs previously selected and we drop the same features (number of nodes, tasks and the submitted hour) to avoid unknown job profile during testing. In a first test run, all the features used to obtain the second columns of Table 2 are the model inputs. In a second test run, the chosen features are only GIDs and task/node, which are the best choices for average power per node prediction found with the IBmodel (hence it keeps same score). We point out to the reader that this favours competing models, especially the ones based on trees for which only the best is retained.

Table 3 presents the results with a range of parameters. It is observed that the IBmodel outperforms all other models when the input space is large. This underlines that there may be no variable selection in the other models. However,

RF and GBRT outperform when we explicitly force the selection of the relevant inputs we have computed previously for all the models. DTR also provides the same results as the IBmodel as it becomes similar to an instance-based model when the dimension is low and the decision tree's leaves may be pure (they can have only one sample in training).

The interpretability of the instance-based models, in particular the selection of explicit features, is a strong advantage as this improves also the other models. Although RF and GBRT are the best performers once the most effective inputs for prediction are known, we do not think they are the most suitable for our application. First, the IBmodel can be updated online whereas RF and GBRT must be completely and regularly retrained in order to handle a job's stream, second, the IBmodel can weight the observations of average power per node of jobs with minimal modification, and finally, it is not easy to explain how RF and GBRT built their predictions.

5.3 Online IBmodel

For practical monitoring of global power consumption through the RJMS it is necessary to provide online and instance weighted models, as already presented in Sect. 4. To demonstrate this claim we compare predictions of future global power consumption available at the submission date of a job by the IBmodel with and without these two improvements. We first construct a reference target with an oracle estimation over time. At any time t, the oracle value is the sum over all running jobs at time t of the jobs' average power consumption. The oracle value is not the true global power consumption as a job's consumption can vary when running, but it is the best approximation following the hypothesis made in [5] and Sect. 2.3 (consumption is constant over the job's entire duration).

With the data from the same period of two months used in the previous section, the IBmodel is trained with only improved weighting (2), only online updating (3) with $\alpha = \exp(\log(0.5)/20)$ (job's contribution is divided by 2 after the 20 next ended jobs with the same profile) and also both (4) with $\tau = 4000$ node hours, then the results are compared to the oracle value of the test set. To compute the estimated value of global power consumption available to SLURM, the list of jobs is converted to a list of events ordered by their time t of three types:

- **Submission event:** The job j is submitted at time t, its average power consumption per node is estimated with the models and buffered for a future event. If models cannot return an estimation (unknown input values), we return the default value 292.89 Watts per node as it is the global average power per node of all the jobs we have.
- **Starting event:** The job j starts at time t, and its average power consumption per node that is estimated at submission is multiplied by the number of nodes to get its power consumption, which is added to the current global power consumption (same for oracle but with true average power per node).

– **Ending event:** Job j ends at time t, then we update the online models and we remove the job's power consumption estimation from the current global power consumption estimation (same with oracle for the latter).

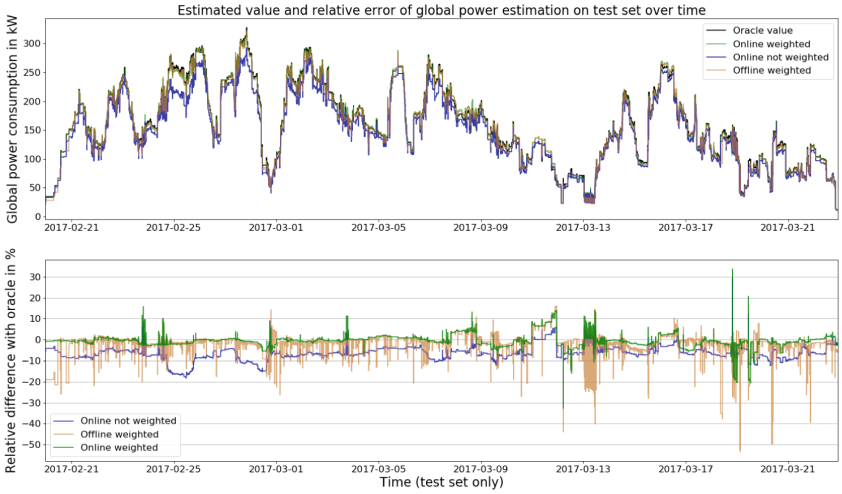

Fig. 4. Up: Evolution of global electrical consumption in Watts over the test period for the offline and online, weighted or not, models. The black curve is the oracle estimation i.e. the evolution's estimate if meanpow of each job is known in advance. **Bottom:** Difference between values predicted by the three models and the oracle

The upper plot of Fig. 4 shows the global power estimation results over time, and the lower plot shows the relative error of the different models compared to the oracle. The online IBmodel (3) without weight adaptation of the jobs underestimates the global consumption given by the oracle by 5% to 10%. The errors of the weighted offline IBmodel (2) peak many times. This suggests that some jobs have profiles that the model did not see enough during training and that they impact the estimation randomly in high proportions. The model needs to be retrained using more recent historical data to improve its estimation, although the spikes will reappear as soon as training stops.

On the contrary, the online and weighted model (4) gives a much more consistent estimation, as the distribution of the relative differences with the oracle are more symmetrical with respect to 0 due to weighting adaptation. The online estimation seems to have stabilized the errors. The peaks in the error patterns may be due to bad default values for new unknown inputs, but as the model is still learning, it only sets a meaningful value for those inputs once a job has ended. Thus, the error remains low even after some time. The absolute deviation of the predictions compared to the oracle is 99% of the time under 12.7 kW, with a mean of 2.40 kW. The relative deviation for 99% of the time is under 10.4%

with a mean of 1.68%. The relative errors approach the measurement precision of the IMPI interface that was used to collect the data.

6 Conclusion

This work shows that it is possible to compute an accurate estimation of the average consumption per node of the submitted jobs using the redundancy of the information provided to SLURM by users. Predictions are computed by an instance-based model which brings several advantages. It is interpretable and shows that jobs on the COBALT computing center have a power consumption that is well predicted by the GID and the number of tasks per node. We show that instance re-weighting and online computations implemented in the IBmodel are necessary to provide a prediction of the global power consumption at submission time that is not underestimated and to stabilize the relative error. The proposed model has a relative error that is of the order of the relative measurement error of the data, which indicates that the IBmodel's performance is already satisfactory.

The next step of this work is to evaluate the capability of the instances model for other computing centers with different behavior of users. This work should be extended by studying the instantaneous power consumption of jobs with time evolving data. Accounting for instantaneous power consumption will allow regulation of each job with a power cap and will enable jobs to be redistributed with more precision.

References

1. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Predictive modeling for job power consumption in HPC systems. In: Kunkel, J.M., Balaji, P., Dongarra, J. (eds.) ISC High Performance 2016. LNCS, vol. 9697, pp. 181–199. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41321-1_10
2. Borghesi, A., Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Scheduling-based power capping in high performance computing systems. Sustain. Comput.: Inform. Syst. **19**, 1–13 (2018)
3. Bugbee, B., Phillips, C., Egan, H., Elmore, R., Gruchalla, K., Purkayastha, A.: Prediction and characterization of application power use in a high-performance computing environment. Stat. Anal. Data Min.: ASA Data Sci. J. **10**(3), 155–165 (2017)
4. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
5. Dutot, P.F., Georgiou, Y., Glesser, D., Lefevre, L., Poquet, M., Rais, I.: Towards energy budget control in HPC. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Press (2017)
6. Eckner, A.: Algorithms for unevenly-spaced time series: moving averages and other rolling operators (2012)
7. Gaussier, E., Glesser, D., Reis, V., Trystram, D.: Improving backfilling by using machine learning to predict running times. In: SC 2015 Proceedings, pp. 1–10. IEEE (2015)

8. Georgiou, Y., Cadeau, T., Glesser, D., Auble, D., Jette, M., Hautreux, M.: Energy accounting and control with SLURM resource and job management system. In: Chatterjee, M., Cao, J., Kothapalli, K., Rajsbaum, S. (eds.) ICDCN 2014. LNCS, vol. 8314, pp. 96–118. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-45249-9_7
9. Holt, C.C.: Forecasting seasonals and trends by exponentially weighted moving averages. Int. J. Forecast. **20**(1), 5–10 (2004)
10. Hunter, J.S.: The exponentially weighted moving average. J. Qual. Technol. **18**(4), 203–210 (1986)
11. Karlin, I., Keasler, J., Neely, J.: Lulesh 2.0 updates and changes. Technical report, Lawrence Livermore National Lab. (LLNL), Livermore, CA, USA (2013)
12. Mitchell, T.M.: The need for biases in learning generalizations (1980)
13. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
14. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
15. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach, 3 Internat. edn. Pearson Education, London (2010)
16. Sîrbu, A., Babaoglu, O.: Power consumption modeling and prediction in a hybrid CPU-GPU-MIC supercomputer. In: Dutot, P.-F., Trystram, D. (eds.) Euro-Par 2016. LNCS, vol. 9833, pp. 117–130. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43659-3_9
17. Sîrbu, A., Babaoglu, O.: A data-driven approach to modeling power consumption for a hybrid supercomputer. Concurr. Comput.: Pract. Exp. **30**(9), e4410 (2018)
18. Storlie, C., Sexton, J., Pakin, S., Lang, M., Reich, B., Rust, W.: Modeling and predicting power consumption of high performance computing jobs. arXiv preprint arXiv:1412.5247 (2014)
19. Tanash, M., Dunn, B., Andresen, D., Hsu, W., Yang, H., Okanlawon, A.: Improving HPC system performance by predicting job resources via supervised machine learning. In: Proceedings of the PEARC, p. 69. ACM (2019)
20. Yamamoto, K., Tsujita, Y., Uno, A.: Classifying jobs and predicting applications in HPC systems. In: Yokota, R., Weiland, M., Keyes, D., Trinitis, C. (eds.) ISC High Performance 2018. LNCS, vol. 10876, pp. 81–99. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92040-5_5
21. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3
22. Zumbach, G., Müller, U.: Operators on inhomogeneous time series. Int. J. Theor. Appl. Financ. **4**(01), 147–177 (2001)