Scheduling with lightweight predictions in power-constrained HPC platforms

Danilo Carastan-Santos, Georges da Costa, Igor Fontana de Nardin, Millian Poquet, Krzysztof Rzadca, Patricia Stolf, Denis Trystram

Scheduling with lightweight predictions in power-constrained HPC platforms

Danilo Carastan-Santos[*] Georges Da Costa[†]
Igor Fontana de Nardin[†] Millian Poquet[†]
Krzysztof Rzadca[‡] Patricia Stolf[†] Denis Trystram[*]

Abstract

With the increase of demand for computing resources and the struggle to provide the necessary energy, power-aware resource management is becoming a major issue for the High-performance computing (HPC) community. Including reliable energy management to a supercomputer's resource and job management system (RJMS) is not an easy task. The energy consumption of jobs is rarely known in advance and the workload of every machine is unique and different from the others.

We argue that the first step towards properly managing power is to deeply understand the power consumption of the workload, which involves predicting the workload power consumption and exploiting it by using smart power-aware scheduling algorithms. Crucial questions are (i) how sophisticated a prediction method needs to be to provide accurate workload power predictions, and (ii) to what point an accurate workload's power prediction translates into efficient power management.

In this work, we proposed a method to predict and exploit HPC workloads power consumption with the objective of reducing the supercomputers power consumption, while maintaining the management (scheduling) performance of the RJMS. Our method exploits workload submission logs with power monitoring data, and relies on a mix of lightweight power prediction methods and a classical EASY Backfillling inspired heuristic. Then, we model and solve the power capping scheduling as a greedy knapsack algorithm. This algorithm improves the Quality of Service and avoids starvation while keeping the solution lightweight.

We base this study on logs of Marconi 100, a 980-node supercomputer. We show using simulation that a lightweight history-based prediction method can provide accurate enough power prediction to improve the energy management of a large scale supercomputer compared to energy-unaware scheduling algorithms. These improvements have no significant negative impact on performance.

[*]University Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG
[†]University of Toulouse, IRIT, CNRS
[‡]University of Warsay, Google

Keywords: Machine learning, HPC, Resource management, Power capping, Simulation

1 Introduction

High-Performance Computing (HPC) technology is becoming more accessible and less expensive to build, which opens the door to new fields to capitalize on the large computational capabilities afforded only by such large systems. However, as opposed to the production cost, the power consumption of HPC platforms only increases, reaching levels [1] in the order of the power consumption of a small city. Besides the carbon footprint issue [2] raised by this increase in the power consumption, current climate events may heavily strain the electricity grids [3] that power HPC platforms. To avoid outages, it has become crucial for HPC platform maintainers to deploy measures to ease the strain in the electricity grid, which is typically achieved by enforcing a power capping over time in the platform. The platform's resource manager must therefore adapt to the available power during this power constrained period.

Most existing works have proposed methods to predict the power consumption of the workload, coupled with a speed scaling (DVFS) method, to adapt to the available power. The drawback is the risk of unforeseeable effects on Quality of Service (QoS). Only few works in the literature have proposed a full framework, including a workload power prediction method feeding energy data at the submission time to a resource manager. These few works often result in complex and/or heavyweight optimization schemes that are perceived to be either too risky that might disrupt regular functioning, or too expensive in terms of computational resources, further reducing the (constrained) power available for the applications.

In contrast with related works, the purpose of this paper is to propose methods that adapt to the available power and deal with the power constraints *as lightweight and simple as possible.* We exploit power consumption data to develop models to predict the power consumption of an HPC application in advance. These models feed power consumption predictions of arriving applications to a scheduler, and the scheduler uses these predictions to comply with the power constraints while keeping the supercomputer operational. Our experimental results highlighted that a lightweight, history-based predictor – which is arguably one of the simplest descriptors of an application power consumption – coupled with an EASY Backfilling [4] inspired scheduler can make a close to optimal use of the available power in constrained periods. We also model the power capping problem as a knapsack problem, solving it using a greedy approach that keeps the scheduling lightweight. This knapsack using the power predictions improves the Quality of Service (QoS) and avoids starvation.

This paper extends our previous work [5] by making the following additions:

1. We added the job's power consumption **standard deviation** to the predicted metrics . The idea is to exploit the central limit theorem to account for the stochasticity of the jobs' power consumption profiles.

2. We introduced a new way to verify if a set of jobs is under power capping using **Gaussian distribution**. Gaussian distribution estimates the probability of being under the power capping using mean and standard deviation predictions for each job. This verification is included in the EASY backfilling scheduling.

3. We modeled the job scheduling under power capping as a **knapsack** problem. To solve this problem, we proposed a greedy approach, maintaining our approach as lightweight as possible. This greedy knapsack can use the same power verification as EASY, such as max, mean, and Gaussian distribution.

4. We largely extended our experimental section to account for the new scheduling algorithms. We perform a deep discussion about the characteristics of each scheduling algorithm. Both EASY and knapsack have been evaluated with Gaussian distribution. We showed that Gaussian verification outperformed our previous best result by better respecting the platform power cap, while keeping a good utilization of the available power. Finally, the greedy knapsack improves the quality of service and avoids starvation by design.

We organized the rest of this paper as follows: Section 2 presents related works in the literature, Section 3 presents preliminary concepts needed to understand our work's context, and Sections 4 and 5 present our methods to predict HPC applications power consumption and to schedule them in an HPC platform, respectively. We present and discuss our experimental results in Section 6, and we present our concluding remarks and future perspectives in Section 7.

2 Related Work

This section provides an overview of the related works regarding supercomputers power monitoring/predictions and prediction-aided HPC resource management. The reader can consult Kocot *et al.* [6] work for a more comprehensive survey on energy-aware resource management in HPC platforms.

Many works have proposed to exploit predictions to improve the performance of HPC resource management. For instance, Zrigui *et al.* [7] used a coarse grain prediction of jobs into long and short to design a scheduling algorithm taking this information into account for the minimization of the maximum completion time of a set of jobs. In [8], the authors proposed a new scheduling algorithm that outperforms the popular EASY backfilling algorithm by 28% considering the average bounded slowdown objective taking into account predictions on the job running times. In the context of predicting the power consumption, Storlie *et al.* [9] developed a framework that predicts energy consumption of arriving jobs. They built a statistical model to approximate the power used by HPC jobs using hierarchical Bayesian modeling with hidden Markov and Dirichlet process models. The goal of their model is to enable the use of an individual

node-capping power strategy shown to be more effective for limiting energy consumption than a uniform one.It is the most wholesome model in comparison to the others, though it comes with a high level of complexity. Bugbee *et al.* [10] proposed another model by combining *a priori* (resource manager's meta-data) and *in situ* data (collected during jobs execution). They focus on the specific applications that exhibit a periodic behavior, which accounts for only 45% of the total workload. Another limitation is that developing fine-tuned models for each possible application may be impractical and too resource demanding. Borghesi *et al.* [11] and more recently Saillant *et al.* [12] and Antici *et al.* [13] proposed Machine Learning (ML) and Rote-Learning approaches that rely on resource manager meta-data in order to predict the power consumption of a HPC workload. They combined this meta-data with measurements from the RAPL [14] interface. Borghesi *et al.* [11] introduced the idea that the mean value is a good descriptor of the HPC applications power consumption.

We distinguish from these works in two aspects: (i) we explored and compared a prediction method that does not rely on ML to predict the power consumption, and (ii) we further investigated on how predicted power consumption statistics can be useful to a resource manager to adapt to the constrained power.

In [15], the authors focused on large-scale parallel jobs for predicting the energy of a parallel application just by observing a few of its active nodes (as opposed to monitor all the deployed nodes). Chapsis *et al.* [16] proposed a power prediction and a resource management framework that includes the power variability due to hardware manufacturing. Their work involves a fine-grained monitoring of the applications and using hardware specific features (hardware counters) to predict the power consumption. Such an approach can be computationally expensive, especially due to the overhead introduced in the computing nodes by the fine-grained monitoring of numerous counters. The approach we proposed in this paper intents to reach a balance in the granularity of the used information: providing coarse grained information on the power profile while using only resource manager related information and past, coarse-grained, monitored executions on the platform.

Our Gaussian method of probabilistic guarantees on power capping is similar to N-sigma method of packing tasks to a single datacenter node [17,18]: however, these works targeted CPU utilization on a single node, while this paper considers power capping the whole HPC machine.

3 Preliminary Concepts

Modern HPC platforms contain large number of nodes. They usually are homogeneous (all computing nodes have the same CPU/accelerators configuration). Many users submit parallel applications (hereafter referred to as jobs) to be executed in the HPC platform, and these jobs can arrive at any point in time (i.e., online job submission). The jobs submissions are managed by the Resources and Jobs Management System (RJMS), such as Slurm [19]. The RJMS decides when and where to process each job. Typical meta-data available to the RJMS

for a given job j is its arrival time (r_j), requested number of processors (q_j) and an estimation of its processing time ($\tilde{p}_j$) which is provided by the user who submitted j. Table 1 presents all notations for this paper. Resources allocation and execution is usually represented in a Gantt chart (Figure 1).

Table 1: Notations

Variable	Description
j	Job
t	Time
s	Sliding window size
w	Week index
$t(w)$	Timestamp of the end of the week w
r_j	Arrival time of job j
q_j	Number of processors of job j
C_j	Completion time of job j
p_j	Processing time of job j
$\tilde{p}_j$	Estimated processing time of job j
Q	Set of jobs waiting in the queue
$\mathcal{J}[t]$	Set of jobs executing in the platform
j	Jobs to start
n	Available processors in the platform
P	Current platform power consumption
$\overline{P}$	Platform power capping
$P_j[t]$	Time-series of the power consumption of job j over the time t
$P_j^{\bullet}$	Observed metric (mean, max, or standard deviation) of job j
$\hat{P}_j^{\bullet}$	Predicted metric (mean, max, or standard deviation) of power consumption of job j
$P_j^{\max}$	Observed max of job j
$\hat{P}_j^{\max}$	Predicted max of power consumption of job j
P_j^{avg}	Observed mean of job j
$\hat{P}_j^{avg}$	Predicted mean of power consumption of job j
P_j^{σ}	Observed standard deviation of job j
$\hat{P}_j^{\sigma}$	Predicted standard deviation of power consumption of job j
W	Jobs considered for the sliding window in weighted average
θ_j	Aging adjustment for jobs of weighted average
α	Aging penalty
$\hat{f}(j)$	Online learning predictor
$\mathbf{J}_{train}$	Dataset of jobs to train the online learning
$\mathbf{J}_{inference}$	Dataset of jobs to predict using the online learning
Kc	Capacity of the knapsack
m	Number of jobs available to put in the knapsack
v_j	Profit of the job j
we_j	Weight of the jobs j
x_j	Boolean indicating whether j is put in the knapsack or not
$wait_j$	Waiting time of the job j at the moment

The RJMS needs even more information when scheduling jobs under power capping. It needs at least a power profile which will serve as a power constraint over a certain time window. Recent HPC platforms are deployed with energy consumption monitoring tools such as IPMI [20], wattmeters, or software modules (often using RAPL [14]). Such an energy monitoring tool can provide power consumption data at the computing node level, as a time series of the power

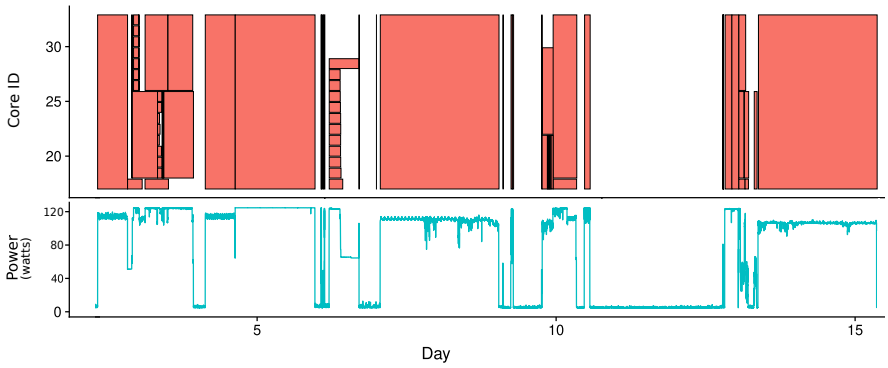


Figure 1: The two sources of data used in a single, 32-core processor example: (top) data coming from the RJMS regarding the jobs execution and allocation (processing time *versus* cores allocated), and (bottom) data coming from a power monitoring tool of the computing node (power *versus* time).

consumption of the computing node (bottom graph in Figure 1).

We can merge the RJMS's jobs data with the computing nodes power monitoring data to get an idea about the power consumption of the jobs. In the case of jobs that share a same computing node (stacked rectangles in Figure 1), the resulting power consumption is in function of each of the jobs energy consumption plus potential interference between the jobs, which makes it hard to distinguish each of the jobs contribution to the nodes power consumption. When taking into account jobs that had exclusive access to computing nodes, however, we can identify a power consumption profile of the jobs. The exact job power profile can only be known after executing the job.

In this work we focused on exploiting the jobs power consumption profiles to perform better scheduling under power constraints *in the simplest way possible*. We choose simplicity because (i) sophisticated methods often lead to numerically complex computations whose power consumption can reduce or at some point nullify the power savings from better scheduling, and (ii) sophisticated methods are often hard to explain/justify which hinders their deployment in practice [21].

More specifically, we focus on the following research questions:

1. Which **simple piece of information** regarding the jobs power profile contribute to better scheduling under power constraints?

2. How to predict this simple piece of information **before the jobs execution**? How much prediction accuracy can we achieve?

3. How to **exploit these predictions** to perform **intelligent jobs scheduling** under power constraints?

We consider as simple piece of information (hereafter referred to as jobs power consumption for simplicity) the **mean**, **maximum**, and **standard deviation** metrics of the jobs power consumption profile. We decide to study the

mean and the maximum because, when used used in isolation, these two metrics can have different impacts in the scheduling with power constraints. We uncovered these impacts in Section 6.5.

4 Predicting the power consumption of HPC jobs

For each job, the RJMS runs the following algorithm before the job's execution: (i) read all the meta-data of the job (including the user's id); (ii) predict statistics of the power consumption of the job (the mean, the maximum and the standard deviation); (iii) provide the job to the scheduler along with power information. The RJMS also has access to past measures, including meta-data and power consumption of past jobs.

We propose two job power prediction methods with an increasing level of complexity to predict the jobs power consumption: the first method uses the power consumption of previous users jobs (users job history). The second method uses previous jobs power consumption data and jobs metadata (Table 2) with Machine Learning (ML) regression methods.

4.1 The first method: predicting power consumption with users jobs history.

For each job j of a given user and and a sliding window size s, we estimate some statistics $\hat{P}_j^{\bullet}$ of its power consumption by taking a weighted average of historical power consumption over previous runs. The statistics we estimate can be the mean ($\hat{P}_j^{avg}$), maximum ($\hat{P}_j^{\max}$), or standard deviation ($\hat{P}_j^{\sigma}$) of the job's power consumption. The general method to predict $\hat{P}_j^{\bullet}$ is defined as follows:

$$\hat{P}_j^{\bullet} = \frac{\sum_{j' \in W} \theta_{j'} P_{j'}^{\bullet}}{\sum_{j' \in W} \theta_{j'}} \tag{1}$$

$$W = \{j' \mid (r_j - s) \leq C_{j'} \leq r_j\} \tag{2}$$

$$\theta_{j'} = \left(1 - \left(\frac{r_j - C_{j'}}{s}\right)\right)^{\alpha} \tag{3}$$

where $C_{j'}$ is the completion time of a previously executed job j' of same user who submitted j, and $P_{j'}^{\bullet}$ is the measured metric of power consumption of j'. The metric of power consumption of a previous job $P_{j'}^{\bullet}$ can be known at the time we predict $\hat{P}_j$ since $C_{j'} \leq r_j$.

Note that this method is job-level-agnostic, i.e., we can use any job-level statistics of power consumption to estimate the same statistics for the next job. More concretely, for each job j', its measured power consumption is a time series, $P_{j'}[t]$, non-zero for t in between the start time and the completion time. So, to estimate the maximum power consumption $\hat{P}_j^{\max}$, we use the measured maximums from previous jobs, replacing $P_{j'}^{\bullet}$ by $P_{j'}^{\max} = \max_t P_{j'}[t]$

in Eq.1. Similarly, to estimate the average power consumption, $\hat{P}_j^{avg}$, we use averages from previous jobs, replacing $P_{j'}^{\bullet}$ by $\hat{P}_{j'}^{avg} = 1/p_{j'} \sum_t P_{j'}[t]$ (where $p_{j'}$ is the duration of the job). Finally, our Gaussian methods additionally use estimated standard deviations, with $\hat{P}_j^{\sigma}$ estimated from standard deviations, replacing $P_{j'}^{\bullet}$ by $\hat{P}_{j'}^{\sigma} = \sqrt{1/p_{j'} \sum (P_{j'}[t] - P_{j'}^{avg})^2)}$ (where $P_{j'}^{avg}$ is the average power consumption; and we take into account only jobs with non-unit duration).

Equation 1 is a weighted average of previous jobs j' of the user that finished within a sliding time window with size s (Equation 2). We assign the weight $\theta_{j'}$ (Equation 3) in function of how long in the past j' finished compared to the arrival time of j. A value of $\theta_{j'} = 0$ means that j' finished at the oldest allowed date, and $\theta_{j'} = 1$ means that the j' finished exactly at the arrival time of j. The parameter α changes the way we penalize older jobs by changing the $\theta_{j'}$ behavior between 0 and 1, from a linear $\alpha = 1$, to a super-linear $\alpha = 2$ or sub-linear $\alpha = 0.5$ fashion.

4.2 The second method: predicting using supervised regression.

The former prediction method cannot harness the jobs metadata (e.g., requested resources, submission time, expected processing time, etc.) to potentially provide better predictions. We can circumvent this problem by using supervised regression with the hypothesis of increasing the prediction accuracy, using the jobs metadata and the power consumption history as input features.

We propose an online learning method to predict the jobs power consumption. The method retrains the prediction models at periodic time intervals (e.g., at the end of each week) in order to adapt, as the jobs history increases and changes. In this context, online still refers to the behavior of the RJMS using all past data, and only meta-data of the arriving jobs. For an *already passed* week with index w (i.e., week 0, 1, 2, ...), let $t(w)$ be the timestamp of when the models will be retrained at the end of week w. Then, we define our job dataset $\mathbf{J}_{train}$ as follows.

$$\mathbf{J}_{train} = \{j \mid C_j < t(w)\} \tag{4}$$

In other words, $\mathbf{J}_{train}$ contains the jobs history. Then, we train a predictor $\hat{f}(j)$ of the jobs power consumption that minimizes the Mean Squared Error (MSE, Equation 5).

$$MSE = \frac{1}{|\mathbf{J}_{train}|} \sum_{j \in \mathbf{J}_{train}} \left(P_j^{\bullet} - \hat{f}(j) \right)^2 \tag{5}$$

After training a predictor $\hat{f}(j)$ we use it to predict the power consumption $\hat{P_{j'}^{\bullet}}$ for all jobs $j' \in \mathbf{J}_{inference}$, where $\mathbf{J}_{inference}$ is defined as follows.

$$\mathbf{J}_{inference} = \{j' \mid t(w) \leq r_{j'} \leq t(w+1)\} \tag{6}$$

In other words, we use the jobs history to train a model at week w that already passed, and use this model to perform predictions of the power consumption of the jobs that will arrive online at week $w+1$. At the end of week $w+1$ (i.e., at timestamp $t(w+1)$) the training procedure repeats, generating a new predictor $\hat{f}(j)$, which will be used for week $w+2$. A particular situation is for week $w=0$, where there is no such dataset to train a regression model. In this case we can use $\hat{f}(j) = \hat{P}_j^{\bullet}$ (Equation 1). We present the choice of regression methods to train $\hat{f}(j)$, and how we exploit the jobs data (i.e, the features of j) in Section 6.2.

5 Scheduling with jobs power profile prediction

5.1 Scheduling algorithms

As the focus of this work is on the impact of power prediction, we will use a classic EASY backfilling algorithm [4]. This algorithm is used in production in a large number of HPC centers. We assume that the platform is static: no failure of nodes nor new nodes are considered during the experiments. We also assume that the servers will always be switched on. Moreover, there is no constraints on the applications related to the host they can run on. All these assumptions enable us to focus on the impact of the prediction framework, further studies on these hypotheses are kept as perspectives.

The implemented EASY uses a first-come first serve (FCFS) policy and works as follows. EASY is executed when the following events occur: a new task arrives or a task finishes. Tasks are stored in a queue in their order of arrival — the order of this queue will then never be changed. EASY starts the oldest jobs in the queue until either finishing the queue or arriving on a job (called high-priority job) that cannot start due to lack of resources. In the later case, EASY will try to *backfill* jobs, that is to say: start waiting jobs right now if they do not delay the highest-priority job estimated starting time.

Algorithm 1 presents the modifications (highlighted in red) of the classic EASY to make it power capping-aware. The main modification we have made to the classical EASY is how to check whether resources are available. Classic EASY only checks whether there are enough available nodes/cores. Here, our implementation named EASY+PC also checks whether there is enough power regarding a given power cap (lines 5 and 11). We assume that an estimator (such as the ones we proposed) can estimate the power needed for a job. Based on this estimator, EASY+PC estimates the currently platform power consumption and the requested power for each job in the queue. All of EASY+PC decisions are based solely on these estimations. Section 5.2 details the methods applied for these verifications.

Besides this EASY+PC, we compare four scheduling algorithms, namely, `EASY FCFS`, `EASY SAF`, `Knapsack Wait.`, and `Knapsack Stretch`. The `EASY FCFS` algorithm is the EASY+PC using FCFS (Algorithm 1), serving as a baseline to compare the other algorithms. `EASY SAF` is similar to Algorithm 1, but

Algorithm 1: EASY+PC backfilling. The highlighted lines show the power verification.

```
   input  : Jobs waiting queue Q, number of available processors n, current
            platform power consumption P, platform power cap P̄
   output: Set of jobs J to start processing
 1 begin
 2     J ← ∅;
 3     Sort Q;
 4     for j′ ∈ Q do
 5         if there are enough resources and power to start j′ then
 6             Update power consumption P;
 7             n ← n − q_j′;
 8             J ← J ∪ j′;
 9         else
10             for j″ ∈ Q \ j′ do
11                 if j″ does not delay j′ and there are enough resources and
                     power to process j″ then
12                     Update power consumption P;
13                     n ← n − q_j″;
14                     J ← J ∪ j′ ;
15             end
16         end
17     end
18 end
```

it sorts the queue using Smallest Area First (SAF) in line 3 [22]. To do so, it calculates the expected processing time multiplied by the number of demanded resources. The jobs with lower area start before. This order helps to reduce the average turnaround by giving high priority to small jobs. However, big jobs may never receive priority, resulting in job starvation [23].

In addition to the two EASY approaches, we modeled our problem as a 0/1 knapsack [24]. This optimization problem considers a knapsack with capacity Kc and m items denoted by index j. Each item j has a profit v_j and a weight we_j. It needs to determine the subset of items that fits into the knapsack and that maximizes the total profit. The classic formulation is to define a boolean x_j that is set to 1 if the item is selected and 0 if it is not [24]:

$$maximize \sum_{j=0}^{j=m-1} x_j \times v_j \tag{7}$$

$$subject\ to \sum_{j=0}^{j=m-1} x_j \times we_j \leq Kc \tag{8}$$

In our case, the items are the jobs in the queue. The capacity Kc is the power capping and the weight we_j is the predicted job power consumption. Regarding the profit, we modeled it as a QoS metric. Since we are trying to improve the turnaround, and the turnaround is highly impacted by the waiting time, we modeled two profit functions using the waiting time. The first one (`Knapsack`

`Wait.`) is the waiting time for every job ($v_j = wait_j$, where $wait_j$ is the waiting time). Thus, the jobs that are longer in the queue receive higher priority, which can be seen as similar to FCFS. The second one (`Knapsack Stretch`) is:

$$v_j = \frac{wait_j + \tilde{p}_j}{\tilde{p}_j} \tag{9}$$

Where $\tilde{p}_j$ is the expected execution time. This formula should increase the priority of the jobs waiting longer, but according to their size. In other words, this functions makes long jobs wait longer than small jobs. However, at some point, these long jobs are prioritized because their priority increases with waiting time (aging mechanism). Therefore, no job can wait indefinitely, avoiding starvation.

A way to solve this knapsack problem is by using dynamic programming. However, the size of our problem explodes the combinatorial possibilities (large number of jobs and large capacity). In addition, our Gaussian approach verifies if we are below the power capping by considering all jobs taken together. For these reasons, we propose a greedy approach to solve our knapsack problem in a reasonable time. Algorithm 2 presents this approach. This greedy algorithm sorts all the jobs by the profit divided by the weight in descending order (line 6). Then, it takes each job in this order and verifies if the job can be executed now. The algorithm stops on the first job that can not fit in the capacity. Outside the power capping window, both greedy knapsacks implement the same algorithm as EASY FCFS since we do not have the power capping anymore (line 3).

Algorithm 2: Greedy knapsack. The highlighted lines show the power verification.

```
   input : Jobs waiting queue Q, number of available processors n, current
           platform power consumption P, platform power cap P̄
   output: Set of jobs J to start processing
 1 begin
 2     if we are outside the power capping window then
 3         Execute Algorithm 1;
 4     else
 5         J ← ∅;
 6         Sort Q by v_j/w_j;
 7         for j ∈ Q do
 8             if there are enough resources and power to start j then
 9                 Update power consumption P;
10                 n ← n − q_j;
11                 J ← J ∪ j;
12             else
13                 break;
14             end
15         end
16     end
17 end
```

5.2 Testing for sufficient power

All the algorithms (in the highlighted lines in Algorithms 1 and 2) above test whether adding a new job j to the set of currently executing jobs $\mathcal{J}[t]$ would violate the platform power cap $\overline{P}$. The test in both Algorithms 1 and 2 consists of two steps: verification and update. Verification (lines 5 and 11 in Algorithm 1 and line 8 in Algorithm 2) tests if placing the new job will violate the power capping $\overline{P}$. If executing the job does not violate the power capping, the update step (lines 6 and 12 in Algorithm 1 and line 9 in Algorithm 2) adds this newly-scheduled jobs to the platform power consumption. We use the following methods for verification and update.

Mean: This method uses the predicted mean of each job. Therefore, assuming that $\hat{P}_j = \hat{P}_j^{avg}$ is the mean, the sum of the means must be lower or equal to the power capping $\overline{P}$. That is: $\hat{P}_j{}^{avg} + \sum_{i \in \mathcal{J}[t]} \hat{P}_i{}^{avg} \leq \overline{P}$. If so, it updates the platform power consumption: $P = \hat{P}_j{}^{avg} + \sum_{i \in \mathcal{J}[t]} \hat{P}_i{}^{avg}$.

Maximum: This method is similar to the previous one, but $\hat{P}_j = \hat{P}_j^{\max}$ is the max instead of the mean. The same verification $\hat{P}_j{}^{\max} + \sum_{i \in \mathcal{J}[t]} \hat{P}_i{}^{\max} \leq \overline{P}$ is applied. If so, it updates the platform power consumption: $P = \hat{P}_j{}^{\max} + \sum_{i \in \mathcal{J}[t]} \hat{P}_i{}^{\max}$.

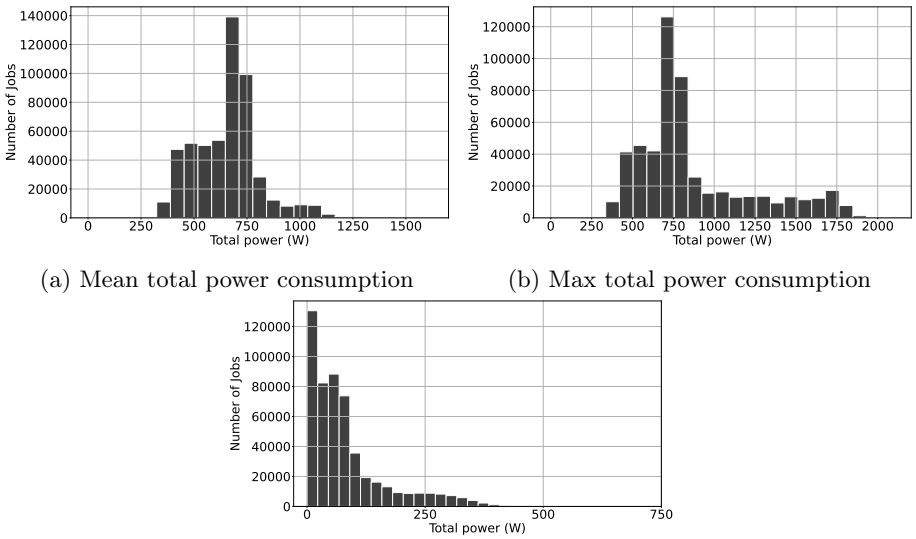
Gaussian: The Gaussian method uses a probabilistic model for jobs energy consumption. This method tries to quantitatively formalize the intuition that, when many jobs execute concurrently, it is unlikely that all jobs simultaneously consume their peak power usage. This approximation uses an estimation of the expected power consumption and an estimation of its standard deviation. Then, as modern platforms execute tens to thousands of jobs concurrently, we use an approximation based on the central limit theorem to convert these estimations into probabilities.

More formally, by linearity of expected value, expected total power requirement $\hat{P}^{avg}(\mathcal{J}[t])$ of the set of jobs executing at time t, $\mathcal{J}[t]$, is simply the sum of expectations over these jobs $\mathcal{J}[t]$, i.e., $\sum_{i \in \mathcal{J}[t]} \hat{P}_i^{avg}$.

For the standard deviation, in general, the variance of the sum of two random variables is the sum of their variances and their covariance, $Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$. However, as jobs in a HPC center are usually started at different times and belong to different users, we will assume that power requirements are not correlated, thus $Cov(X, Y) = 0$. This leads to a simple estimator of a standard deviation of a set of jobs as: $\hat{\sigma}(\mathcal{J}[t]) = \sqrt{\sum_{i \in \mathcal{J}[t]} (P_i^{\sigma})^2}$.

When testing whether an extra job J fits the power budgets, we extend these estimates with the estimates for this job, getting the expected value: $\mu = \hat{P}_j{}^{avg} + \hat{P}^{avg}((J)[t])$ and a standard deviation of: $\sigma = \sqrt{(\hat{P}_j{}^{\sigma})^2 + \sum_{i \in \mathcal{J}[t]} (\hat{P}_j{}^{\sigma})^2}$.

We can now replace the crisp fitting test with a probabilistic one: if $\mu + \sigma < \overline{P}$, the set of jobs stays within the energy budget with an approximate probability of 0.68. If $\mu + 2\sigma < \overline{P}$, the set of jobs stays within the energy budget with an approximate probability of 0.95. And, similarly, if $\mu + 3\sigma < \overline{P}$,



(a) Mean total power consumption

(b) Max total power consumption

(c) Power consumption standard deviation

Figure 2: Distributions of the actual mean, maximum, and standard deviation of power consumption per node of the filtered jobs in the Marconi100 trace.

this probability is approximately 0.997.

6 Results

6.1 Jobs power prediction: dataset description

This work uses the trace collected from the Marconi100 supercomputer [25]. Marconi100 consisted of 980 computing nodes, each of which consisted of a two-socket IBM POWER9 AC922 (32 cores in total), and four NVIDIA Volta V100 GPUs. The trace contains jobs meta-data such as jobs submission times, processing times, anonymized user ids, and node ids allocated to the jobs. It also contains data about the nodes' total power consumption, measured at each 20-second periods, using an IPMI module installed in the nodes.

From this trace we used the data regarding the operation of the Marconi100 from January 2022 to September 2022. To circumvent the limitation mentioned in Section 3, we filtered out the jobs that shared a node from the original trace. We also filtered out jobs that run in less than a minute because they have too few power measurements. After filtering we end up with a dataset of 523,204 jobs submitted by 576 users.

Figure 2 shows the mean, max, and standard deviation power consumption of the computing nodes for each of the jobs in the dataset. We can observe a peak density of jobs with mean and max consumption of around 700 watts. This insight in itself could serve as a prediction if the whole distribution was clear

Table 2: Features used in the mean power consumption regression methods

Feature	Description
1	Submission time (hour of the day)
2	Number of processors (q_j)
3	Number of nodes
4	Requested processing time $\tilde{p}_j$
5	The sliding window history prediction $\hat{P}_j^{\bullet}$ (Equation 1)
6	Standard deviation $\sigma(\{P_{j'} \mid j' \in W\})$
7	The power consumption $P_{j'}$ where j' is the last finished user job

and concentrated at around 700 watts. However, the jobs distribution is not so clear for other power values. which justifies the need of more sophisticated prediction methods.

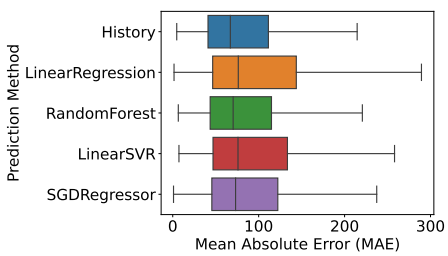
6.2 Jobs power prediction: experimental setting

For the history based power prediction method (Section 4), we set the parameter α (Equation 3) with a value of 2. This choice is based on the hypothesis that recent jobs have more importance than older jobs when predicting the power consumption. An $\alpha = 2$ puts more weight into recent jobs, and the weight decreases fast for older jobs. We set the sliding window size s to account for the whole user's job history, which translates to s being completion time of the first finished job of a user. We decided on this design to avoid eventual empty sliding windows during the prediction process. For the regression based power prediction method, we use the features presented in Table 2. Features 1 to 4 are standard job data that can be obtained at job submission, and features 5 to 7 are lagged features (i.e., a standard feature engineering technique), which can be obtained by using the user's job submission history.

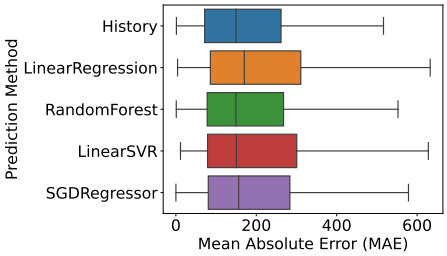
We trained predictors $\hat{f}(j)$ using a selection of regression methods in the `scikit-learn` [26] library: (i) Linear Regression (`LinearRegression`), (ii) Random Forest (`RandomForestRegressor`), (iii) Support Vector Regression with Linear Kernel (`LinearSVR`), and (iv) Stochastic Gradient Descent Regression (`SGDRegressor`). For each training week and method, we applied `scikit-learn`'s recursive feature elimination technique to choose the appropriate subset of features from Table 2 to use according to the training data. We perform this feature elimination and we set each of the regression methods hyper-parameters with a 5-fold cross validation scheme.

6.3 How to predict jobs power information before their execution? How much prediction accuracy can we achieve?

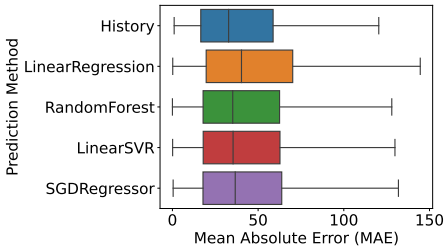
Figure 3 shows the mean absolute error (MAE) of the methods used to predict the mean and the max jobs power consumption. The boxplots represent the distribution of the prediction performance for each of the 576 users present in our job dataset. For the mean power, we achieved a median prediction MAE from 67 W (using jobs history method, Section 4) to 76.2 W (using linear regression, Section 4). For predicting the maximum power consumption, we achieved a median prediction MAE from 149.3 W (jobs history method) to 170.1 W (linear regression). Lastly, for the standard deviation, we achieved a median prediction MAE from 32.7 W (jobs history method) to 40.2 W (linear regression). For all power metrics (i.e., mean, maximum and standard deviation) we can not clearly distinguish which prediction method is better. We can observe, however, that our jobs history prediction method achieves equivalent prediction performance than the more sophisticated ML methods.



(a) Mean total power consumption



(b) Max total power consumption



(c) Total power consumption standard deviation

Figure 3: Mean Absolute Error of the mean, maximum, and standard deviation power consumption prediction methods for the jobs in the Marconi100 trace.

This is an important finding. Because we develop these prediction methods to control the power consumption of operating supercomputers, we must reduce the energy consumption overhead induced by introducing these methods as much as possible. Achieving a high level of power prediction performance

with the lowest level of overhead is therefore a priority. Although all of the regression methods used can be seen as lightweight when compared to neural network methods, a simple jobs history based method is clearly much less energy demanding than a Machine Learning regression method, which requires much more processing steps (i.e., normalizing data, selecting the best features and finding the best hyper-parameters with cross-validation, etc.).

6.4 Jobs scheduling with power prediction: experimental setup

We simulate 30 different workloads using EASY+PC (Section 5) with various powercap values and with various job power predictors. Each workload consists of jobs taken from the filtered Marconi100 supercomputer dataset (Section 6.1). Jobs are selected following workload trace replay guidelines [27]. EASY+PC applies a power constraint solely during the first 3 hours of each simulation (power capping window). The powercap values we use range from 40% to 70% of the highest dynamic power consumption of the Marconi100 dataset in 2022 (955080 W).

We study EASY+PC's behavior depending on the information it uses to determine the power consumption of a job:

- predicted `mean`, predicted `max`, and three Gaussian distributions (`gaussian_68`, `gaussian_95`, and `gaussian_99`) using predicted mean and standard deviation;
- real `mean`, real `max`, and three Gaussian distributions (`gaussian_68`, `gaussian_95`, and `gaussian_99`) using real mean and standard deviation;
- `naive` uses the maximum reachable power consumption of a job (i.e., 2100W * q_j, where 2100W is the maximum single-node power value present in the Marconi100 dataset in 2022). `naive` is used as a baseline predictor to evaluate the accuracy of the other predictors proposed.

We use Batsim [28] and SimGrid [29] to perform the scheduling simulations, using a SimGrid representation of the whole 980-node of Marconi100. We use the schedule produced by the simulators plus the time-series data about the jobs power consumption from the Marconi100 dataset to calculate the total power consumption. Taking into account the 30 workloads and the job power predictors, our experimental campaign consists of 1320 simulation instances. Additionally, each workload is executed on EASY (without powercap) to serve as baseline and evaluate both impacts on total power consumption and QoS.

6.5 Which jobs power information contribute to better scheduling under power constraints?

Figures 4 and 5 summarize our simulation campaign (Section 6.4) by aggregating data from all workloads. One workload has been excluded from Figure 5 as

powercapping greatly increases scheduling performance on it. Values given in the remaining of this section are computed by (workload, powercap) group, and then averaged on all groups. They analyze (I) the extent to which each predictor is able to use the power at its disposal while powercap is active, and (II) how they degrade QoS performance (through turnaround time) compared to baseline EASY's.

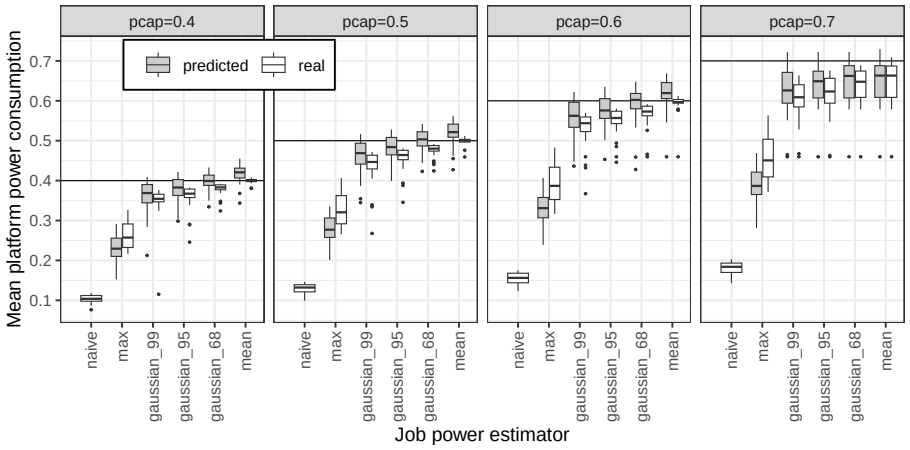


Figure 4: Distribution of the platform power consumption during the powercap-constrained 3-hour time window. The power cap is the horizontal black line. All power values (y axis, facet powercap `pcap`) are expressed as a proportion of the maximum dynamic power range. Here and in the remaining plots we use standard boxplots, with boxes spanning between the first and the third quartiles, and whiskers reaching the furthers outliers within 1.5 of the interquartile range.

Since `naive` considers the maximum achievable node power for the whole jobs duration, EASY+PC is incapable of harnessing the power consumption fluctuations that occur during job execution to better use the available power. This incapability leads to a severe power under-utilization (74%), and a significant increase in the turnaround time (14%).

`max` provides better information about the maximum power consumption than `naive`, thus better harnessing the fluctuations. `max` still remains, however, as a "conservative" method which hypothesizes that the maximum value occurs all the time during the jobs execution. Such hypothesis helps assuring that EASY+PC does not trespass the power cap (it never did, even when using our `max` prediction method), though this results in power under-utilization and increase in the turnaround time (respectively around 44% and 8%).

More "aggressive", `mean` hypothesizes that the mean power is the value that occurs most of the time during the jobs execution. This hypothesis gives more flexibility to EASY+PC to harness the jobs power fluctuations. The drawback is the increased risk of exceeding the powercap. In our experiments, the `mean` method is the one that makes the best use of the available power (0.96% power

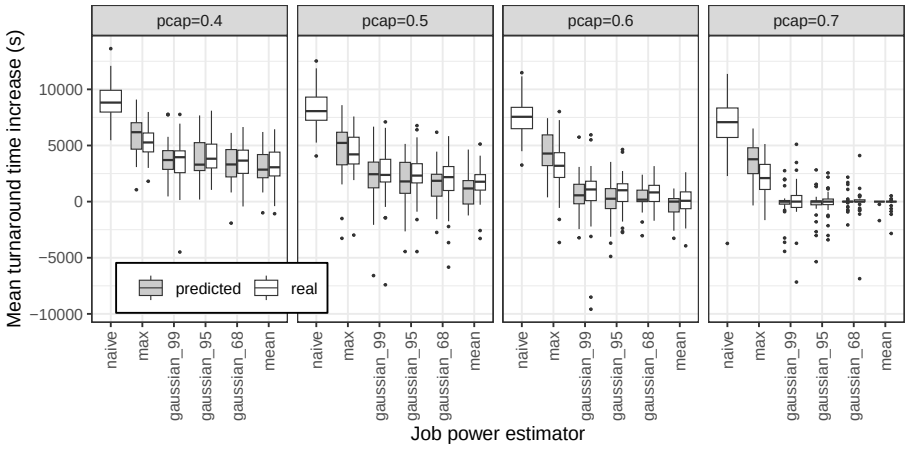


Figure 5: Distribution of the performance degradation (compared to EASY) for 29/30 workloads. The turnaround time of a job is the amount of time the job spends in the system (from submission to finish). The workload mean turnaround time is the arithmetic mean all the jobs turnaround time. Standard boxplots.

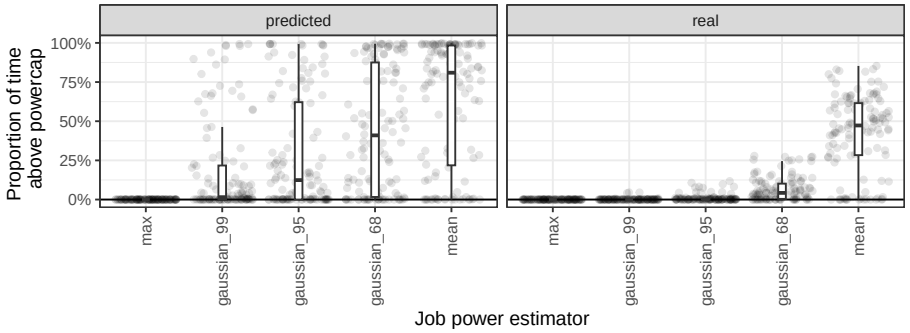


Figure 6: The proportion of the time above the power cap for the 30 workloads and the four power capping methods using predicted and real power consumption. This value goes from 0% (no violation) to 100% (during the entire 3-hour simulation window, the power consumption is above the cap). A dot represents a single instance. The lower, the better.

over-utilization in average, but 3% when ignoring the power capping 0.7) and the one that increases the turnaround time the least (1.65%). Using `mean` trespasses the powercap in most instances (88%) but the trespassing is small: the maximum instantaneous powercap break observed is in average and median 11% above the powercap. Figure 6 shows how much time each execution stays above the power capping for each way to verify the power consumption. This Figure shows that

the `mean` has several cases where the power usage stays above the power capping for a long period, even using the real mean value for each job. Therefore, even if the break is not high, it keeps violating the power capping for a good part of the window.

Aiming to reduce the aggressiveness of the `mean` approach, we introduce the three Gaussian ways to verify the power capping (as explained in Section 5). Among them, `gaussian_68` is the most aggressive, and `gaussian_99` is the most conservative. They all present a mean under-utilization (8%, 4%, and 0.7% for `gaussian_99`, `gaussian_95`, and `gaussian_68`, respectively) and a turnaround increase close to the mean approach (around 2% for all of them). Concerning the maximum instantaneous power capping break observed, they reduce it to 3%, 5%, and 8% for `gaussian_99`, `gaussian_95`, and `gaussian_68`, respectively (while `mean` has 11%). Regarding the proportion of the time above the power capping (Figure 6), we can see that all Gaussian approaches reduce this proportion compared to the mean, with the `gaussian_99` being the best among the Gaussian.

Lastly, the prediction accuracy of the mean and standard deviation methods (Section 6.3) results in satisfactory performances when compared to using real values. It impacts both the `mean` and Gaussian approaches. Please note that real values are baselines and cannot be obtained before the jobs execution.

6.6 How the predictors can be introduced to other scheduling algorithms?

After presenting the power prediction results, we implement four scheduling algorithms to provide other possibilities for using our predictions. The main idea is to improve the Quality of Service (QoS) of the results, using the predictions to stay below the power capping. We analyze the same workloads as the previous sections under a power capping of 0.5. Regarding the power verification, we used for all algorithms the same `gaussian_99` as before. In this section, we compare the four algorithms presented in Section 5: `EASY FCFS`, `EASY SAF`, `Knapsack Wait.`, and `Knapsack Stretch`. Figure 7 presents the results for these algorithms.

First, it is possible to notice that the three new algorithms (`EASY SAF` and the two knapsacks) are more aggressive than `EASY FCFS` considering the power capping (Figure 7a). The reason is that they run more jobs inside the power capping window (see Figure 7d). `EASY FCFS` has 18% of the total number of jobs inside the window, and the others have more than 30%. Since they run more jobs, they are more sensitive to mean and standard deviation variations from the prediction. Taking the results using the real mean and standard deviation, both knapsacks are very close to the power capping. The reason is that these algorithms tend to maximize the items inside the "knapsack", approximating the sum weight of the items to the max knapsack capacity. Nevertheless, this behavior increases the probability of violating the power capping linked to the prediction errors (see the predicted results on Figure 7a). Regarding the QoS, Figure 7b shows that `EASY FCFS` has the worst mean turnaround time increase,

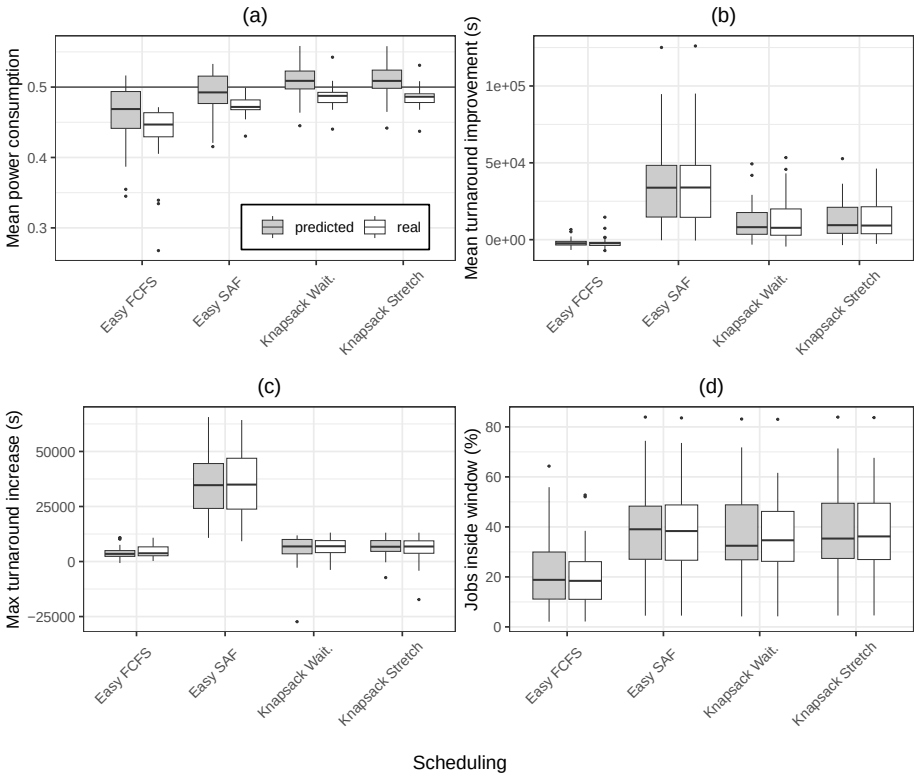


Figure 7: The results of different scheduling policies using real mean and standard deviation, and `gaussian_99` power capping verification. This figure is divided into four parts: (a) The mean platform power consumption, highlighting how far the algorithms are to the power capping (the closer to the line, the better); (b) the mean turnaround time improvement compared to a EASY FCFS without power capping, showing the impact on the QoS. The negative values of EASY FCFS means that it is worst than the the baseline (the higher, the better); (c) the max turnaround time increase compared to a EASY FCFS without power capping, highlighting the starvation of the job with worst turnaround (the lower, the better); and (d) the percentage of the workload jobs (in number) executed inside the power capping window (the higher, the better).

and `EASY SAF` has the best. `EASY SAF` gives priority to small jobs, which helps to reduce the impact of waiting time on them. However, Figure 7c illustrates the main `EASY SAF` drawback. This algorithm increases the large jobs waiting time making these jobs starve. On the other hand, both knapsack algorithms improve the mean turnaround compared to the `EASY FCFS` without starving large jobs. Comparing both knapsacks, they have quite similar results, with the `Knapsack Stretch` having slightly better results considering the QoS.

7 Conclusion

We presented in this paper three main contributions: a complete integrated environment from monitored data to the jobs execution, an evaluation of several jobs power consumption prediction methods, and some scheduling algorithms to use the predictions in the decision making process. In particular, we showed that "lightweight" (frugal) predictions used in the scheduling module lead to similar performance improvements, when compared to more costly and sophisticated predictions or compared to the optimal value. Simple history prediction method (such as mean and standard deviation) is sufficiently good to express the jobs power profiles during scheduling, which fosters lower-cost scheduling algorithms.

The proposed approach focused on the capability to take into account these lightweight predictors for a classical and widely used EASY scheduling policy. In particular, we proposed a knapsack algorithm for determining a good trade-off between performance and the power limits. From this positive experience on EASY and knapsack, the next step is to investigate more complex scheduling policies harnessing the new information from the predictors, but also adding actual monitoring values to improve the quality of its decision. It would also be interesting to refine the jobs model, to take into account phases of long duration applications. Using such information would help to improve the management of short duration jobs.

Acknowledgements

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This work was supported by the research program on Edge Intelligence of the Multi-disciplinary Institute on Artificial Intelligence MIAI at Grenoble Alpes (ANR-19-P3IA-0003), ENERGUMEN (ANR-18-CE25-0008), the France 2030 NumPEx Exa-SofT (ANR-22-EXNU-0003) and Cloud CareCloud (ANR-23-PECL-0003) projects managed by the French National Research Agency (ANR), REGALE (H2020-JTI-EuroHPC-2019-1 agreement n. 956560), and LIGHTAIDGE (HORIZON-MSCA-2022-PF-01 agreement n. 101107953). A CC-BY public copyright licence has been applied by the authors to the present document and will be applied to all subsequent versions up to the Author Accepted Manuscript arising from this submission, in accordance with the grants' open access conditions. We thank Salah Zrigui for starting the study on the job energy profiles. We also thank Francesco Antici for curating and sharing the Marconi100 dataset.

Artifact Availability

The experiments described in this article have been made with open science and reproducibility concerns in mind. Code, data and documentation to reproduce our work is available on Zenodo [30].

Conflicts of Interest

Krzysztof Rzadca is also affiliated with Google.

References

[1] "Frontier's architecture," https://www.olcf.ornl.gov/wp-content/uploads/Frontier-Architecture-Overview_Abraham.pdf, 2024, last access 18 October 2024.

[2] N. Bates, G. Ghatikar, G. Abdulla, G. A. Koenig, S. Bhalachandra, M. Sheikhalishahi, T. Patki, B. Rountree, and S. Poole, "Electrical grid and supercomputing centers: An investigative analysis of emerging opportunities and challenges," *Informatik-Spektrum*, vol. 38, no. 2, pp. 111–127, 2015.

[3] Wikipedia, "2021 Texas power crisis, Online; last access 29 november 2023," https://en.wikipedia.org/wiki/2021_Texas_power_crisis, 2023.

[4] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the ibm sp2 with backfilling," in *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*. IEEE, 1998, pp. 542–546.

[5] D. Carastan-Santos, G. Da Costa, M. Poquet, P. Stolf, and D. Trystram, "Light-weight prediction for improving energy consumption in hpc platforms," in *Euro-Par 2024: Parallel Processing*, J. Carretero, S. Shende, J. Garcia-Blas, I. Brandic, K. Olcoz, and M. Schreiber, Eds. Cham: Springer Nature Switzerland, 2024, pp. 152–165.

[6] B. Kocot, P. Czarnul, and J. Proficz, "Energy-aware scheduling for high-performance computing systems: A survey," *Energies*, vol. 16, no. 2, p. 890, 2023.

[7] S. Zrigui, R. Y. de Camargo, A. Legrand, and D. Trystram, "Improving the performance of batch schedulers using online job runtime classification," *Journal of Parallel and Distributed Computing*, vol. 164, pp. 83–95, 2022.

[8] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015.

[9] C. Storlie, J. Sexton, S. Pakin, M. Lang, B. Reich, and W. Rust, "Modeling and predicting power consumption of high performance computing jobs," 2015.

[10] B. Bugbee, C. Phillips, H. Egan, R. Elmore, K. Gruchalla, and A. Purkayastha, "Prediction and characterization of application power use in a high-performance computing environment," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 10, no. 3, pp. 155–165, 2017.

[11] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "Predictive modeling for job power consumption in hpc systems," in *High Performance Computing*, J. M. Kunkel, P. Balaji, and J. Dongarra, Eds. Cham: Springer International Publishing, 2016, pp. 181–199.

[12] T. Saillant, J.-C. Weill, and M. Mougeot, "Predicting job power consumption based on rjms submission data in hpc systems," in *High Performance Computing*, P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, Eds. Cham: Springer International Publishing, 2020, pp. 63–82.

[13] F. Antici, K. Yamamoto, J. Domke, and Z. Kiziltan, "Augmenting ml-based predictive modelling with nlp to forecast a job's power consumption," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1820–1830.

[14] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, mar 2018.

[15] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode, "Predicting the energy and power consumption of strong and weak scaling hpc applications," *Supercomputing Frontiers and Innovations*, vol. 1, no. 2, 2014.

[16] D. Chasapis, M. Moretó, M. Schulz, B. Rountree, M. Valero, and M. Casas, "Power efficient job scheduling by predicting the impact of processor manufacturing variability," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 296–307.

[17] N. Bashir, N. Deng, K. Rzadca, D. Irwin, S. Kodak, and R. Jnagal, "Take it to the limit: peak prediction-driven resource overcommitment in datacenters," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 556–573.

[18] P. Janus and K. Rzadca, "Slo-aware colocation of data center tasks based on instantaneous processor requirements," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 256–268.

[19] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *JSSPP 2003*. Springer, 2003, pp. 44–60.

[20] G. Da Costa, J.-M. Pierson, and L. Fontoura-Cupertino, "Mastering system and power measures for servers in datacenter," *Sustainable Computing: Informatics and Systems*, vol. 15, pp. 28–38, 2017.

[21] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *JSSPP: IPPS'97 Processing Workshop Geneva, Switzerland, April 5, 1997 Proceedings 3*. Springer, 1997, pp. 1–34.

[22] D. Carastan-Santos, R. Y. De Camargo, D. Trystram, and S. Zrigui, "One can only gain by replacing easy backfilling: A simple scheduling policies case study," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 1–10.

[23] A. Tanenbaum, *Modern operating systems*. Pearson Education, Inc.,, 2009.

[24] S. Martello and P. Toth, "Algorithms for knapsack problems," *North-Holland Mathematics Studies*, vol. 132, pp. 213–257, 1987.

[25] A. Borghesi, C. Di Santi, M. Molan, M. S. Ardebili, A. Mauri, M. Guarrasi, D. Galetti, M. Cestari, F. Barchi, L. Benini *et al.*, "M100 exadata: a data collection campaign on the cineca's marconi100 tier-0 supercomputer," *Scientific Data*, vol. 10, no. 1, p. 288, 2023.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] J. Emeras, "Workload Traces Analysis and Replay in Large Scale Distributed Systems," Theses, Université de Grenoble, Oct. 2013.

[28] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator," in *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States, May 2016. [Online]. Available: https://hal.science/hal-01333471

[29] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014.

[30] M. Poquet, D. Carastan-Santos, I. Fontana de Nardin, G. Da Costa, K. Rzadca, P. Stolf, and D. Trystram, "Artifact data of article "Scheduling with lightweight predictions in power-constrained HPC platforms", TPDS 2024," 2024. [Online]. Available: https://doi.org/10.5281/zenodo.13961003