Estimating the job's pending time on a High-Performance Computing cluster through a hierarchical data-driven methodology

Fabio Carfí
fabio.carfi@polito.it
Department of Control and Computer Engineering,
Politecnico di Torino
Turin, Italy

Enrica Capitelli
enrica.capitelli@external.cnhind.com
CNH Industrial
Turin, Italy

Vladi Massimo Nosenzo
vladi.nosenzo@cnhind.com
CNH Industrial
Turin, Italy

Tania Cerquitelli
tania.cerquitelli@polito.it
Department of Control and Computer Engineering,
Politecnico di Torino
Turin, Italy

ABSTRACT

Nowadays, manufacturing industries have to shorten the time to market to satisfy customers' needs and survive in globally competitive contexts. For these reasons, all the steps required to create new products need to be optimized as much as possible to minimize the overall execution time. Technologies like CAD (Computer-Aided Design) and CAE (Computer-Aided Engineering) are useful in this process. At the same time, Finite Element Methods are fundamental because they allow performing simulations, not only of mechanical components but also of an entire manufacturing process. Doing so, there is the possibility to avoid errors which would entail the need to carry out further operations and, therefore, increase execution time during the product's creation. In the context of heavy simulations performed on High-Performance Computing (HPC) architectures, it is challenging to estimate each job's pending time, i.e., the time between the job submission and the job execution starting time. This information could support the optimization of the resource requests, drastically reducing the waiting times needed to obtain the simulations' results. This paper presents an innovative and hierarchical data-driven methodology to estimate the discretized value of pending time, i.e., estimating the range time that a job will have to wait before the necessary hardware resources are supplied for its execution. A large set of experiments have been performed on a real dataset collected in an Italian industrial context to assess the proposed approach's effectiveness in correctly and accurately predict the pending time's discretized value. This work is a preliminary step towards implementing a scheduling system based on machine learning techniques.

KEYWORDS

Data-driven methodology, HPC cluster, Hierarchical Classification Approach

1 INTRODUCTION

Nowadays, manufacturing industries make extensive usage of physical HPC clusters[8] or online data centers to execute a huge amount of simulations for mechanical components or entire manufacturing processes. In this way, it is possible to slightly reduce incoming errors related to the final product, trying to shorten the time needed for its creation. The problem is that these simulations request lots of resources that are often not immediately available, increasing the time needed to start the job execution and, consequently, obtaining the final results. The waiting time, but also the execution time, can also be affected by hardware, software, node, and many other types of failures that can occur, wasting cluster resources [11, 24]. Generally, many simulations are needed for a single product and, for this reason, there is the necessity to overcome these problems. The topic is significant in the real context of software applications development in the industrial field. This requires efficient and innovative solutions that can benefit from data analytic and machine learning techniques. Some research studies have addressed this research issues.
The available data to be analyzed are characterized by a strong class imbalance (both in the categorical and continuous context) because most of the submitted jobs have limited waiting times, while only a limited part of them request high waiting times. The estimation of these times is challenging because it is influenced by different factors (like, for example, the computation type, the parameters used to run the simulations, the available resources' usage, and many others) not known a priori.
First, we consider a set of features describing the job submission's context. Some of them were available within the database provided (like the amount of resources requested to run the job, the software to use, and the submit, start and end times available for each job). We will refer to them as the *accounting data*. In contrast, other attributes related to the HPC cluster context (like the number of jobs in the queue and the number of running jobs at the submit time) have been identified using a feature selection process, detailed in Section 3.2, to which we will refer to as *context data*. It is also essential to notice that developing an effective data-driven methodology would create a strong impact in using HPC's hardware resources, whose optimization would significantly reduce software testing costs. The implemented solution belongs to the hierarchical classification and it is composed by three classification levels, each one characterized by thresholds (for the definition of the local class of prediction) and training records that depends on the results of the previous level.
The paper is organized as follows. Section 2 illustrates the related work concerning the optimization of resources allocation in the HPC context. In Section 3 a brief description of the data is given, discussing the cleaning and feature engineering operations used

to build the dataset, avoiding as many outliers as possible, and also depicting the solution implemented, describing all the main parts that compose it and the most important features provided. Section 4 discusses the experiments executed during the implementation process. In fact, the proposed methodology has been validated on a real-data set collected in the context of a CNH (an American-Italian UK-headquartered multinational corporation) HPC cluster, proving the effectiveness and accuracy of the methodology in making correct predictions. Finally, in Section 5, we conclude the paper by proposing some future improvements of the proposed data-driven methodology towards implementing a scheduling system based on machine learning techniques.

2 RELATED WORK

Some studies have addressed fundamental aspects for the improvement of resources allocation. In fact, the scientific literature has focused on two main problems: (i) the job failure prediction and (ii) the implementation of scheduling algorithms based on machine learning techniques.
The first approach aims to predict how a specific job will end its execution, estimating if an error will occur or the results will be returned. The implemented algorithms try to stop those jobs whose termination status is predicted as a failure [9, 11, 12], independently of its type. The study of unsuccessful jobs and tasks execution can improve the performance and the energy saving of an entire HPC cluster [10, 19], but request lots of data to be used to define a pattern between the attributes and the target variable. The most significant variables can be available from the start, saved in different databases, or extracted from the available data with some feature engineering processes. Anyway, it is important to notice how parsing and transformation operations [6] are extremely useful to obtain better prediction results. Some failures are not so easy to predict because they are rare, meaning that they occur fewer times than the other, but can anyway produce lots of waste of energy and resources [14]. For example, Liu et al. [11] implemented a system for the failure prediction composed of two main algorithms: the first one is a job clustering algorithm used to measure the correlation among jobs with a various number of tasks. In contrast, the second one is a multitask learning algorithm used to get similar information from different correlated jobs. In this paper, however, we do not address this problem because the data available are not sufficient to achieve reliable results.
A parallel research approach has been devoted to studying how a scheduler can optimize the available resources. Jassas et al. [9] also suggested, after the analysis of finished and failed jobs behavior, the development of scheduling algorithms to improve the reliability and availability of cloud applications. In large-scale HPC, a waiting queue is needed when the system is highly employed, whereas the number of resources cannot be unlimited. With their studies, Nurmi et al. [15] have confirmed that, despite the prediction efficiency of a resources scheduler, the performance is also affected by the job's waiting time that increase when there are limited resources.
Among the open issues in the research field of industrial software development, the design of a data-driven HPC cluster scheduler to optimize the resources allocation is challenging. To this aim, this paper proposes a data-driven methodology for estimating a job's pending time by combining other techniques available in the state-of-the-art appropriately. Like Park [16], we study, design, and develop a data-driven model to improve the scheduler performance by analyzing the jobs log file. Unlike Park, we derive a prediction model estimating the uncertain time of the submitted job.

3 THE PROPOSED METHODOLOGY

Here we present our data-driven methodology to derive a predictive model that estimates the pending time of submitted jobs. The proposed methodology follows the KDD (Knowledge Discovery from Database) main steps [23], starting from the analysis of the available data up to the deployment of a model able to predict the time interval of the target variable with good performance.

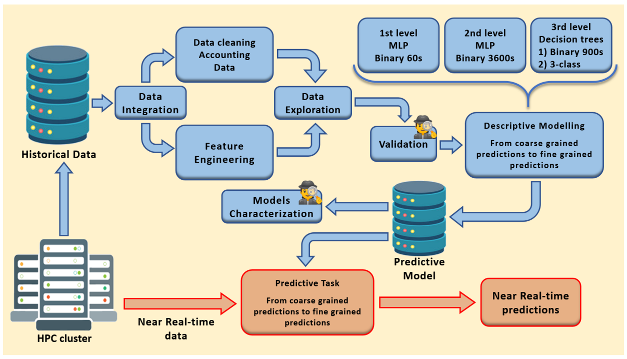

Historical Data
Data Integration
Data cleaning Accounting Data
Feature Engineering
Data Exploration
1st level MLP Binary 60s
2nd level MLP Binary 3600s
3rd level Decision trees 1) Binary 900s 2) 3-class
Validation
Descriptive Modelling From coarse grained predictions to fine grained predictions
Models Characterization
Predictive Model
HPC cluster
Near Real-time data
Predictive Task From coarse grained predictions to fine grained predictions
Near Real-time predictions

Figure 1: Schema of the proposed methodology

Figure 1 shows the main building blocks of the proposed methodology. Given the small amount of data available, Data Integration, Data Cleaning, Feature Engineering, and Data Exploration operations have been carried out, which are extremely useful respectively for organizing and cleaning up the data and then perform statistical analyses to get a broader view of the problem. Then, we moved on to the experimental phase, called Validation in the schema reported in Figure 1, where several tests were carried out to understand the behavior of machine learning algorithms. Thanks to this phase, we arrived at implementing the model later presented in Section 3.4.

3.1 Use case description

The monitoring system running on CNH for a job submitted to be executed on the cluster collects some types of data, including:

1) *Accounting data* that contain all the information about the job submitted like the submit, start and end times, the number of resources requested, the identifier of the user that submitted it, and many others, for a total of about twenty attributes.
2) *HPC queues' technical specifications.*
3) *Context data* that have been computed using the information from the previous two categories and containing those attributes related to the HPC cluster context at the time of the job submission.

Among all the variables within the accounting data table, the attributes concerning the number of requested resources, like the number of CPUs requested or the software to use for the job execution, were highlighted as useful, along with the times related to the execution itself (submit, start and end of the job), while many others were discarded, due to a low utility for the model learning process (like in the case of the user identifier) or the high presence of missing values. On the other hand, the majority of the context attributes were selected for this prediction

problem. The cardinality of accounting and context data is about 25 thousand records, after the cleaning process.
Given the difficulty of the problem, a specific analysis of the available data was made so that the prediction categories could be defined. Based on the results of the data exploration step and the discussion with the domain experts, the classes of interest were defined as follows: negligible or not (identified by a 60-seconds threshold), greater or less than one hour, or other sub-ranges of the last step. Through these three levels, it is possible to support the user in deciding whether or not to submit the job at that specific moment.

3.2 Data preparation and exploration

Some cleaning operations were necessary to remove possible errors within the accounting data. The cleaning process involved: 1) the elimination of all those records that have not started their execution (run time equal to 0), indicating the presence of hardware problems that invalidate the execution of the job, 2) the deletion of variables that do not provide any useful information such as the alphanumeric identifier of the user that submitted it, 3) the removal of all the duplicate jobs present inside the dataset, as well as 4) the deletion of those records characterized by ambiguous attribute values that cannot be inferred or computed. Besides, we search for a status variable associated with each job, found inside PBS (Portable Batch System, the simulations manager) raw files, reporting possible software problems during the submission of the job and all those records with a value that differs from 0 were deleted. Carrying out the cleaning operations, the data available went from 105 thousand records to just over 25 thousand, minimizing the possibility of prediction errors caused by external conditions.
The above-mentioned dataset does not contain any reference to the status of the HPC cluster at the submission time, data that would be extremely useful for predicting the waiting time of a job. For this reason, using the information regarding both queue on which the job is launched and the submit, start, and end times associated with its execution, a feature engineering process has been carried out to extrapolate the attributes that allow characterizing the HPC context at the submit time of each available record. Almost all the available data were used, applying only a selection of the data cleaning operations previously analyzed, as jobs that cannot be useful for the prediction process hold anyway the resources of the HPC cluster and therefore conditioned the times of other jobs submitted later. After the entire extraction process, we selected only the records with the information related to those jobs that were not affected by the accounting data's cleaning operations. With this feature engineering process, we have extracted, for each tuple, the number of jobs that, at the submission time, were waiting for the necessary resources, the number of those that, instead, was in execution, the total waiting and execution time up to that moment, the number of cores and nodes available at the submission time.
The data preparation is a semi-supervised process that has been executed manually, step by step. As usual, the pre-processing is a custom pipeline requiring a strong interaction with the domain experts to continually assess the selected data's quality and remove noisy data. It includes different steps deeply dependent on the key aspects of available data and the feedback given by domain experts to assess each cleaning step's correctness and the quality of the obtained results. The proposed strategy strongly depends on available data, and it is not easy to generalize the procedure to be used in a different setting.
Finally, after obtaining the necessary information, exploratory analyses were carried out on both accounting and context data to extract as much knowledge as possible to use for the prediction algorithm. We studied mainly the distribution of variables, through tests such as Shapiro-Wilk [20] and graphical visualizations like bar charts or box plots [13], and also the type of relationship between the variables, using correlation matrices, generated with the Spearman correlation coefficient [22], and tests such as χ^2(Chi-squared) and η^2(Eta-squared). More specifically, we studied the distribution of the target variable for the accounting data through some clustering operations that have highlighted the high presence of values in the low part of the range. About the correlation matrix of the context data, reported in Figure 2, we looked at the cells containing values greater than 0.85 or lower than -0.85 because, in this case, only one of the two attributes should be considered, being the two highly statistically related. Besides, the distribution analysis has been deepened for accounting data, also performing descriptive analysis and clustering operations.

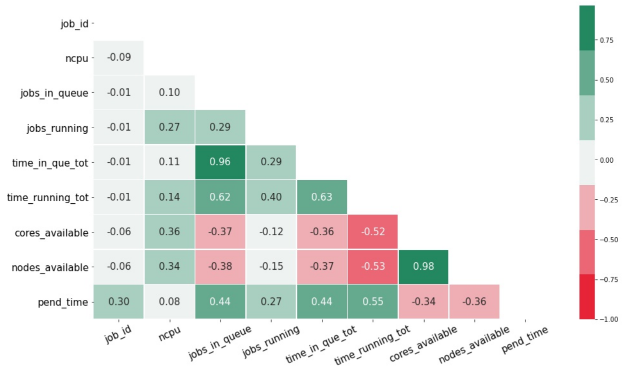


Figure 2: Correlation matrix generated for the context data

3.3 Data labelling

Here we discuss the distribution of the target variable, by means of different statistical methods (see Figure 3, 4 and 5), and explain the decision taken to define the prediction classes.
Figure 3 shows the frequency of pending time value of every job available in the real dataset, highlighting for each classification level, represented in Figure 3a, 3b and 3c, the thresholds applied. It is important to notice that as levels increase, the classification is more specific and ranges are more limited. The x-axis have a logarithmic scale to represent all the data available, while the y-axis have a linear scale in which the absolute frequencies of each value are reported. The higher frequencies are observed for values less than 60 seconds, noting that around this value, the curve tends to flatten on frequencies close to 0. This because, for values higher than this threshold, as also shown by the graph in Figure 4, there is a low number of jobs compared to those belonging to the first prediction class. On the other side, for values greater than 60 seconds the variability of the attribute increases, as shown by the low frequencies displayed in Figure 3. The records' distribution in the different classes shows, in fact, a clear imbalance between class 0 and the others, being the first class cardinality eight or more times higher than the other classes. However, this can be considered acceptable in a real context since, for times less than

a minute, the wait can be considered negligible, highlighting a good CNH HPC cluster sizing.

Figure 5 shows the distribution of the target variable for each specific prediction class. From this representation a highlighted aspect is the presence of outliers within the clusters at the extremes, which means that boxplots are crushed on the lowest values of the graph. On the other hand, for the other clusters, one information that can be derived is that most of the values belonging to each cluster tend to concentrate around the median. From the analysis of the boxplots, in addition to the dispersion of the data around the median, it is also clear that there is a higher medium-low values frequency within the clusters, which explain the downward displacement of the box. This confirms the existence of a positive asymmetry, as already highlighted by the bar chart shown in Figure 4.

It is possible to say, that each one of the three classification levels of the implemented system has a specific objective:

- The first level's objective is to identify those jobs that have a negligible waiting time.
- The second level points to provide feedback to the user, giving him the ability to decide whether to wait to submit the job, considering that the threshold is an hour.
- The third manages more critical situations, specializing the results obtained by the second level.

The choice of thresholds' values for each classification level is experimentally-driven, meaning that it is based entirely on the results of experiments previously conducted, the results of which are discussed in Section 4. The identification of the classes to be predicted, in terms of ranges for each of them, for the first level has been defined with a data-driven approach. An iterative process based on classification was used, with several classes ranging from 20 to 2. The search ended in correspondence with the best results, identified by a binary classification with a 60-seconds threshold. This one admits distinguishing between negligible times and the most significative ones firstly. In fact, with a time less than 60 seconds, every time can be considered as acceptable. The values obtained by all the experiments (see Section 4) are reported in Table 5, 6, 7 and 8. From the evaluation of the models, which results are reported in Table 1, we decided to use the MLP (Multi-level Perceptron) [7] for the prediction of the first classification level.

Considering that the results for class 0 are extremely meaningful, with a 94% of correct predictions, we decided to apply the other levels only to class 1. Considering the low data availability (about six thousand records, i.e., the ones correctly classified in the first step), we decided to perform another binary classification. The classes thresholds have been set after performing different experiment to find a good trade-off between the prediction reliability and the number of associated records for each specific class. After finding a compromise by using a 3, 600-seconds threshold, we studied the behaviour of all the algorithms, obtaining the results reported in Table 2, which highlight the suitability of the MLP as classifier of the second level.

We decided to apply a further classification to both of the second level classes for the third level. For the first one, whose range goes from 60 to 3, 600 seconds, considering that the range is quite narrow, we opted for a binary classification with a 900-seconds threshold, thus creating classes 1.0.0 and 1.0.1, sufficient to give users fairly precise feedback. To decide which model was the most accurate in the assignment of records belonging to class 1.0, an evaluation of the algorithms has been carried out, obtaining the values shown in Table 3, from which we derived that the Decision Tree [3] was the one that better fits the situation. For the classification applied, instead, for the class 1.1 of the second level, considering that the range is wide (from 3, 600 seconds up to a few days of waiting), a binary classification was not the proper approach. From the multi-class experiments previously executed (discussed in Section 4.2), we noted that the one with higher results was a 3-class classification, which shows also a great improvement compared to the other experiments (taking into account the results reported in Table 6). In this context, the 7, 200- and 10, 800-seconds thresholds adopted to create classes 1.1.0, 1.1.1 and 1.1.2 were defined to meet some conditions: 1) do not compromise too much the performance and 2) obtaining classes to return meaningful feedback to the users. After identifying these fundamental characteristics, we examined all the selected algorithms and, from the results obtained and reported in Table 3, it was possible to highlight that the Decision Tree was the one that better fits the situation.

3.4 Data modeling

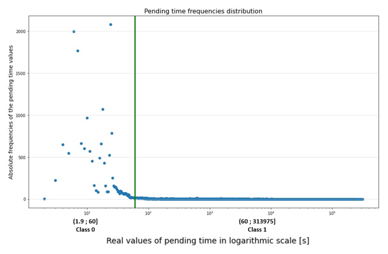
The solution found is part of the category known as hierarchical classification [21] (a specific case of ensemble learning [17]) in which more algorithms are used to predict the belonging class. In fact, the system created, whose schema is shown in Figure 6, is composed of several prediction levels in which the algorithms' training depends on the class predicted in the previous level. Only correctly classified records are reused for the algorithm's training in later levels, resulting in a physiological decrease of records each time you move to a following level.

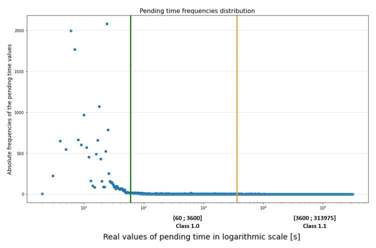
The algorithms that make up the system are the MLP and the Decision Tree. The first one belongs to Neural Networks. The final model comprises several interconnected layers, each composed of many neurons, where each connection has a weight associated. These algorithms' learning process is based on the adaptation of the weights to minimize the difference between the real value and the predicted one. These are good algorithms for real-time predictions but are characterized by a high initialization time. On the other hand, Decision Trees are non-parametric algorithms able to predict the target variable value through simple conditions inferred from the data. The final model is made up of nodes (specific variables), branches (possible values of the variables), and leaves (final class of prediction). This algorithm is easy to visualize and understand why a specific value is returned but is low-generalizable. Even a small variation of a variable value is enough to change the final result completely.

In Figure 6 we try to schematize the operation of the hierarchical process with which the best results have been obtained. In essence, all the steps that the system performs to assign the most appropriate class are depicted, given the input variables provided. As it is possible to see in Figure 6, we refer to the first level ranges as Class 0 and 1, to the second level as Class 1.0 and 1.1 and to the third level as Class 1.0.0, 1.0.1, 1.1.0, 1.1.1 and 1.1.2.

3.5 Model characterization

It is important to note that all the algorithms that make up the hierarchical model must be interpretable so that, starting from specific input values, it is clear the process for which a certain value is returned. This is implicit for the Decision Trees used in the third level. They are algorithms that can be interpreted by definition, giving the possibility to draw the entire rules' tree. Instead, in the case of the two MLPs used for the classification


(a) First classification level


(b) Second classification level

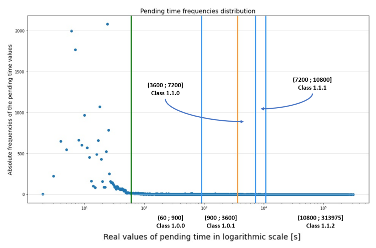

(c) Third classification level

Figure 3: Representations of the records distribution considering the belonging class and the classification level

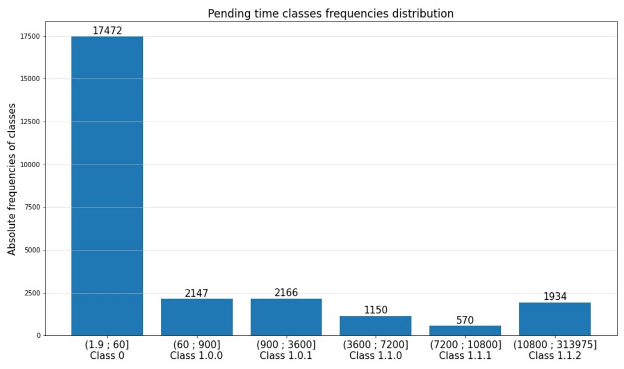


Figure 4: Distribution of records for each prediction class

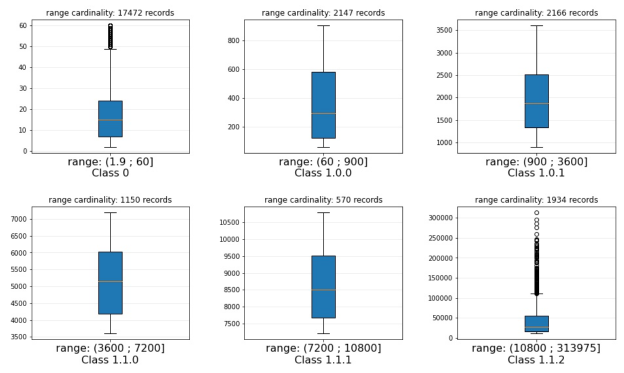


Figure 5: Box-plot, for each prediction class, of the records' distribution

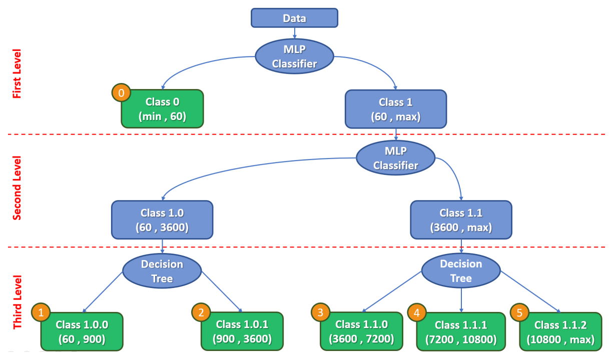


Figure 6: Schema of the best classification solution found

of the first and second level, being these considered as black box algorithms, it was necessary an intermediate step of interpretability with the use of Decision Trees: the latter is provided with the input variables of each MLP, while as target variable the predictions made by the MLP are used. This process is outlined in Figure 7.

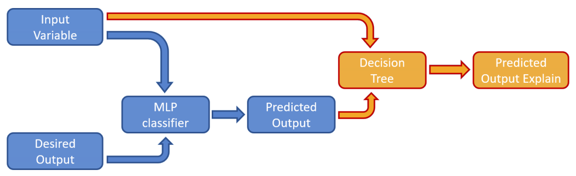


Figure 7: Schema of the explanation process of the MLPs through Decision Trees

Using these tests, the results obtained are extremely accurate since the Decision Tree employed for the first level explains the MLP to 99%, while the one used in the second level to 93%.

3.6 Data prediction

Every time a new job arrives, the first level MLP is used to classify the record. If the classification result is 0, the prediction system stops, meaning that this job will wait at most 60 seconds before entering in execution. Differently, if the result of the classification is 1, the system passes to the second level of the system, that will perform a binary classification again with an MLP classifier, and then to the third one, depending on the class predicted in the second level, to better understand the range to which the real pending time belongs. In fact, if the result of the second level is 0, in the third level is performed a binary classification, while, if the result is 1 will be performed a 3-class classification, in both cases using a Decision Tree classifier. The different approach adopted for the two third-level predictions derives from the difference between the classes 1.0 and 1.1 (the first from one minute to one hour, while the second ranges from more than one hour to a few days of waiting), even if the number of training records is quite the same.

3.7 Model evaluation

For the evaluation of every algorithm used in all the performed experiments, subsequently reported in Section 4, we employed two different strategies: (i) for the regression experiments (see Section 4.2) we used the *k-fold cross validation*, (ii) while for the classification ones we used the *stratified k-fold cross validation*. Both these strategies can obtain reliable estimates of the algorithms' performance when they work with unseen data, finding an acceptable bias-variance trade-off [18]. These techniques are used to assure the robustness of the models' results. This section also defines how and when we used them, giving details on the k-parameter used in the different experiment types and the final solution implemented.

The k-fold cross validation is a well-known state-of-the-art strategy widely used in the machine learning and data mining community to estimate the performance of supervised algorithms, assessing the validity of the experiments' results. It is composed by some simple steps:

1) Split the dataset into k disjoined groups (subsets)
2) For each subset:
   - Take the group as test set
   - Take the remaining $k - 1$ groups as training set
   - Fit a model on the training set and evaluate it on the test set
   - Retain the evaluation and discard the model
3) The model performance is estimate as the average of the k experiments executed

The stratified k-fold is a particular case of the k-fold cross validation, which retains, in each group, the same label distribution of the entire dataset.
By using these two strategies we can ensure that the presented results, reported in Section 4, are robustly evaluated, minimizing the possibility of occurring errors.
The k value is selected considering the cardinality and the type of the available data, but, generally, 10 is a good standard value. For the regression experiments we used a k-fold cross validation with $k = 10$ and also with $k = 20$ to evaluate the models, while for the classification ones we adopted a stratified k-fold cross validation with $k = 20$ for the majority of them. For the hierarchical experiments, we used both $k = 20$ and $k = 30$, considering the variability of the attributes and the data cardinality for each classification level. Because of that, in the proposed solution, analysed in Section 3.4, we used a stratified k-fold with $k = 20$ for the first two levels, while for the last one we applied this strategy using $k = 30$.

4 PRELIMINARY EXPERIMENTAL VALIDATION

Here we discuss the experimental results performed to address two precise objectives:

1) Evaluating the performance of the different selected algorithms for the implementation of the proposed solution levels (see Section 4.1)
2) The experimental comparison of the proposed solution with respect to the state-of-the-art techniques (see Section 4.2)

4.1 Performance evaluation

Here we discuss the performance evaluation of the proposed methodology. The first prediction level with a 60-seconds threshold has been defined as two classes, whose cardinalities are 17, 472 records (jobs) for the first one and the remaining 7, 967 for the second one. From the evaluation of all the selected algorithms, using a stratified 20-fold cross-validation [23], we obtained the results reported in Table 1. Looking at these results, the most accurate algorithms are the KNN (k-Nearest Neighbors) [1] and the MLP. Considering the high initialization time for both algorithms, the MLP allows a faster prediction in real-time, even for very complex networks. For this reason, it was chosen for the first classification level.
To train the second prediction level, we used only the records that in the first level were correctly classified as belonging to class 1. In this case, we used a binary classification with a threshold set to

Table 1: Results for the first level classification of the best classification approach

Algorithm	Mean Accuracy	Precision Class 0	Precision Class 1	Recall Class 0	Recall Class 1	F-measure Class 0	F-measure Class 1
SVM Classifier	0.895515	0.879496	0.948320	0.976763	0.704782	0.925581	0.808612
KNN Classifier	0.908408	0.899230	0.935290	0.976019	0.760136	**0.936052**	**0.838665**
Decision Tree	0.904910	0.894905	0.934796	0.976190	0.748588	0.933782	0.831393
Random Forest	0.902551	0.891687	0.935556	0.976763	0.739802	0.932288	0.826243
MLP Classifier	0.908959	0.896712	0.945873	0.980369	0.752353	**0.936676**	**0.838087**

Table 2: Results for the second level classification of the best classification approach

Algorithm	Mean Accuracy	Precision Class 1.0	Precision Class 1.1	Recall Class 1.0	Recall Class 1.1	F-measure Class 1.0	F-measure Class 1.1
SVM Classifier	0.673340	0.634847	0.718535	0.725892	0.626304	0.677324	0.669257
KNN Classifier	0.648482	0.613054	0.689112	0.693395	0.608283	0.650754	0.646180
Decision Tree	0.676510	0.656513	0.695347	0.662310	0.689851	0.659398	0.692588
Random Forest	0.679847	0.662974	0.694618	0.655245	0.701865	0.659087	0.698223
MLP Classifier	0.707040	0.671890	0.745378	0.742141	0.675624	**0.705270**	**0.708789**

3, 600 seconds. Re-evaluating the algorithms with a stratified 20-fold cross-validation, the results, shown in Table 2, were obtained. From these values, it is evident that the MLP is the algorithm with the best performance values among all, the reason why it was chosen as a classifier for the second level.
Considering the results of the MLP, the algorithm that best fits the situation, 4, 238 records have been used for the training of the third level that are equivalent to those correctly predicted both as class 1.0 and as 1.1, respectively 2, 101 for the first one (about 46.5% belonging to class 1.0.0 and the remaining 53.5% to class 1.0.1) and the remaining 2, 137 for the second one (about 20% belonging to class 1.1.0, 15.5% to class 1.1.1 and the remaining 64.5% to class 1.1.2). For the third classification applied to class 1.0, it was decided to perform a binary classification using a 900-seconds threshold, thus creating the two final classes, 1.0.0 and 1.0.1. To evaluate the algorithms in this situation, we used a stratified 30-fold cross-validation, obtaining the results reported in the second and third columns of the Table 3. From these values, it is possible to notice that all algorithms predict better just one of the two classes, all except the Decision Tree that predicts both classes well and better than all the other models. Finally, for the third level classification applied to class 1.1, it was decided to perform a 3-class classification using 7, 200- and 10, 800-seconds thresholds, thus creating classes 1.1.0, 1.1.1, and 1.1.2. To evaluate the algorithms' performance in the prediction of these 3 classes, a stratified 30-fold cross-validation was performed, obtaining the results reported in the last four columns of Table 3. From these data, we firstly discarded the SVM (Support Vector Machine) [5] and KNN algorithms, due to the null results in class 1.1.1. Between the other three models, to decide which one was the best for this step, we decided to take into account the average of the results of the three classes, since all three have the same importance, resulting in that the Decision Tree was also in this case, the one with better predictions' performance.

4.2 Comparative analysis with state-of-the-art approaches

We compared the solution with some kind of state-of-the-art techniques to prove the effectiveness of the results obtained:

- *Case* 1: regression techniques.
- *Case* 2: single level classification techniques with balanced classes.

Table 3: Results for the third level classification of the best classification approach

Algorithm	F-measure Class 1.0.0 (1)	F-measure Class 1.0.1 (2)	F-measure Class 1.1.0 (3)	F-measure Class 1.1.1 (4)	F-measure Class 1.1.2 (5)	Mean F-measure (3 + 4 + 5)
SVM Classifier	0.657624	0.529112	0.261386	0	0.795812	0.352399
KNN Classifier	0.519897	0.590207	0.288288	0.075631	0.757982	0.373967
Decision Tree	**0.644612**	**0.639501**	**0.386555**	**0.233261**	**0.795609**	**0.471808**
Random Forest	0.537203	0.619853	0.386684	0.250922	0.767198	0.468268
MLP Classifier	0.604927	0.569154	0.336232	0.234514	0.796296	0.455680

- *Case* 3: single level classification techniques where the classes were manually defined.
- *Case* 4: single level binary classifications with manually defined thresholds.
- *Case* 5: single level binary classifications with thresholds defined by summing ranges from *Case* 2 and *Case* 3 experiments.
- *Case* 6: hierarchical classification experiments with two classification levels.
- *Case* 7: hierarchical classification experiment with three classification levels.

About the *Case* 1, the proposed solution has been initially compared with a reference baseline given by regression algorithms' usage, considering that the target variable is a numeric real value. In fact, we executed some k-fold cross-validation experiments, using firstly $k = 10$ and then $k = 20$, and also a Holdout. From the results of the cross-validation experiments, reported in Table 4a and 4b, we can see that there is an improvement of the max accuracy, meaning that some folders are better to predict than others, while the mean accuracy decrease, meaning that there are more other folders with no acceptable results. Same as for the holdout, which results, reported in Table 4c, are too low to be considered. Established that this kind of algorithm does not achieve good performance, we moved on to classification experiments.
The classification experiments can be grouped into two main types: single level (*Case* 2, 3, 4 and 5), meaning that it is used only one algorithm, and multi-level (*Case* 6 and 7), i.e., using a hierarchical structure. We started with a single level where the classes were defined in such a way as to have classes of the same frequency. Unlikely, for *Case* 2, noticeable from the results reported in the Table 5, independently of the number of classes, the accuracy increases too slowly (looking the results from Table 5a to Table 5d), reason why this type of experiments was deemed unsuccessful.
For this reason, we moved to *Case* 3 experiments, manually defining the classes to be predicted, depending on the feedback we would return and on the information obtained from the previous tests. Anyway, also from these results, reported in Table 6, we can highlight that the F-measure increase too slowly, looking at the results of Table 6a, 6b and 6c, while, for the last experiments, the results represented in Table 6d show a clean improvement. For this reason, only the information of the last experiment was used for the creation of the ranges for the binary experiment that made up the first level of the solution proposed with a 60-seconds threshold.
Considering the non-relevant results with multi-class experiments, we passed to *Case* 4 experiments and we started to study algorithms' behaviour with binary classifications using, firstly, three user-defined thresholds: 900, 1, 800, and 3, 600 seconds. From the values obtained, reported in Table 7, it is possible to see that as the threshold increase, the results of class 1 tend to

Table 4: Results obtained from the regression experiments

(a) 10-fold cross validation

Algorithm	Max Accuracy	Mean Accuracy
Polynomial Regressor	0.532658	0.235069
KNN Regressor	0.400554	0.172238
Decision Tree (depth=3)	0.453951	0.270740
Decision Tree (depth=5)	0.493425	0.177651
MLP Regressor	**0.532969**	**0.325323**

(b) 20-fold cross validation

Algorithm	Max Accuracy	Mean Accuracy
Polynomial Regressor	0.551733	0.006580
KNN Regressor	0.581727	0.080216
Decision Tree (depth=3)	0.533837	0.200991
Decision Tree (depth=5)	0.571285	0.173368
MLP Regressor	**0.630281**	**0.220024**

(c) Holdout

Algorithm	Accuracy
Polynomial Regressor	0.217887
KNN Regressor	0.119690
Decision Tree (depth=3)	0.199433
Decision Tree (depth=5)	0.149165
MLP Regressor	**0.233375**

Table 5: Results obtained from the classification experiments with same frequency classes

(a) 20 classes

Algorithm	Mean F-measure
SVM Classifier	0.120080
KNN classifier	0.143910
Decision Tree	0.177636
Random Forest	0.170767
MLP Classifier	**0.179786**

(b) 10 classes

Algorithm	Mean F-measure
SVM Classifier	0.250410
KNN classifier	0.230243
Decision Tree	0.272162
Random Forest	0.265845
MLP Classifier	**0.281455**

(c) 7 classes

Algorithm	Mean F-measure
SVM Classifier	0.335620
KNN classifier	0.318225
Decision Tree	0.365250
Random Forest	0.370206
MLP Classifier	**0.385226**

(d) 4 classes

Algorithm	Mean F-measure
SVM Classifier	0.426678
KNN classifier	0.419915
Decision Tree	0.461583
Random Forest	0.465651
MLP Classifier	**0.470602**

decrease, related to a decrease of the number of records associated with that class, while the results of class 0 do not change significantly.
For this reason, we defined new thresholds belonging to *Case* 5 experiments: 24, 30 and 582 seconds. These have been defined by the aggregation of ranges of multi-class experiments previously executed, considering their specific confusion matrices. In this case, we can notice an increase in both classes results, reported in Table 8, increasing the threshold used. The 60-seconds threshold used inside the solution proposed as the first prediction level, discussed in Section 3.4, also belongs to this type of experiments.

Table 6: Results obtained from the classification experiments with user defined classes

(a) 7 classes

Algorithm	Mean F-measure
SVM Classifier	0.204707
KNN classifier	0.275410
Decision Tree	0.319689
Random Forest	**0.323275**
MLP Classifier	0.305662

(b) 6 classes

Algorithm	Mean F-measure
SVM Classifier	0.365455
KNN classifier	0.323907
Decision Tree	0.385265
Random Forest	0.380419
MLP Classifier	**0.390795**

(c) 5 classes

Algorithm	Mean F-measure
SVM Classifier	0.317228
KNN classifier	0.378983
Decision Tree	**0.444375**
Random Forest	0.436247
MLP Classifier	0.436944

(d) 3 classes

Algorithm	Mean F-measure
SVM Classifier	0.632459
KNN classifier	0.637632
Decision Tree	0.706892
Random Forest	0.673945
MLP Classifier	**0.721240**

Table 7: Results obtained from the binary classifications with user defined thresholds

(a) 900 seconds threshold

Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.933264	0.735641
KNN classifier	0.930409	0.756369
Decision Tree	**0.935545**	**0.775042**
Random Forest	0.932919	0.757683
MLP Classifier	0.932859	0.756791

(b) 1800 seconds threshold

Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.940621	0.711091
KNN classifier	0.921904	0.668386
Decision Tree	0.937039	0.704289
Random Forest	0.932726	0.694544
MLP Classifier	**0.940739**	**0.720261**

(c) 3600 seconds threshold

Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.954124	0.675447
KNN classifier	0.927332	0.529040
Decision Tree	0.949024	0.670578
Random Forest	0.941362	0.631338
MLP Classifier	**0.951396**	**0.684249**

We discuss now the multi-level experiments, starting with two hierarchical experiments, the first with four and the second with five classes, both with two predictive levels (*Case* 6 experiments). The schemas of these two structures are represented in Figure 8a and 8b. The results obtained, reported in the table 9a and 9b, anyway, highlight a worsening of the F-measure values globally. In fact, if we consider the range up to 900 seconds, we can see that the mean F-measure of the classes in this range, for both these two levels of hierarchical structures, is worse compared with the solution discussed in Section 3.4. The only positive thing that can be highlighted in the system shown in Figure 8b is the slightly better result regarding the 3-class classification compared to the proposed solution, however at the cost of a great worsening of the performance in the preceding ranges. For these reasons, both the experiments have been discarded.

Table 8: Results obtained from the binary classifications thresholds deducted from previous experiments

(a) 24 seconds threshold

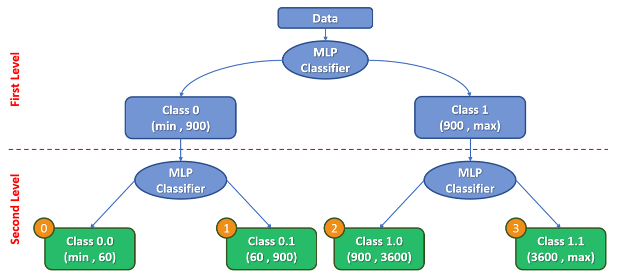
Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.842161	0.685678
KNN classifier	0.844086	0.723856
Decision Tree	0.848624	0.726922
Random Forest	0.837363	0.716383
MLP Classifier	**0.850326**	**0.725032**

(b) 30 seconds threshold

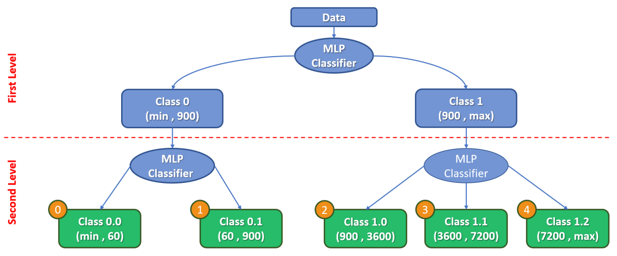
Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.885184	0.722498
KNN classifier	0.897168	0.781996
Decision Tree	**0.904586**	**0.791383**
Random Forest	0.900460	0.784943
MLP Classifier	0.904360	0.790468

(c) 582 seconds threshold

Algorithm	F-measure Class 0	F-measure Class 1
SVM Classifier	0.925858	0.727657
KNN classifier	0.932483	0.795663
Decision Tree	**0.934467**	**0.796841**
Random Forest	0.928620	0.763013
MLP Classifier	0.933949	0.789122



(a) Schema of the classification experiment with 2 levels and 4 final classes



(b) Schema of the classification experiment with 2 levels and 5 final classes

Figure 8: Hierarchical experiments with two levels schemas

Table 9: Results obtained from the hierarchical classification experiments with 2 prediction levels

(a) 2 levels and 4 classes

Algorithm	F-measure Class 0.0	F-measure Class 0.1	F-measure Class 1.0	F-measure Class 1.1
SVM Classifier	0.967544	0.281269	0.412197	0.841287
KNN classifier	0.969164	0.440555	0.431903	0.803104
Decision Tree	0.969975	0.428648	0.469654	0.816419
Random Forest	0.968557	0.403009	0.497378	0.809666
MLP Classifier	**0.970665**	**0.414648**	**0.491721**	**0.840145**

(b) 2 levels and 5 classes

Algorithm	F-measure Class 0.0	F-measure Class 0.1	Mean F-measure (3-classes step)
SVM Classifier	0.967544	0.281269	0.434300
KNN classifier	0.969164	0.440555	0.455720
Decision Tree	0.969975	0.428648	0.506964
Random Forest	0.968557	0.403009	0.496920
MLP Classifier	**0.970665**	**0.414648**	**0.522985**

Table 10: Results obtained from the hierarchical classification experiment with 3 prediction levels and 6 final classes

Algorithm	F-measure Class 0.0	F-measure Class 0.1	F-measure Class 1.0	Mean F-measure (3-classes step)
SVM Classifier	0.967544	0.281269	0.412197	0.323296
KNN classifier	0.969164	0.440555	0.431903	0.386362
Decision Tree	0.969975	0.428648	0.469654	0.435682
Random Forest	0.968557	0.403009	0.497378	**0.452004**
MLP Classifier	**0.970665**	**0.414648**	**0.491721**	0.438479

Finally, we made a comparison with a hierarchical architecture with three levels and six classes (*Case* 7 experiment), whose schema is reported in Figure 9, based on the two levels hierarchical experiments discussed previously. In fact, we added a third level to better predict the time range from 3, 600 seconds onward. The obtained performance is reported in Table 10, and they are not so good since even the results of the 3-class classification are worse than the solution proposed in section 3.4.

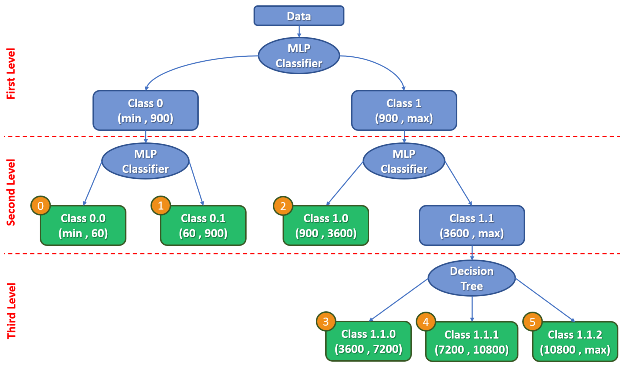


Figure 9: Schema of the classification experiment with 3 levels and 6 final classes

5 CONCLUSION AND FUTURE RESEARCH DIRECTION

This paper proposed a data-driven solution to predict, with good accuracy, the interval in which a submitted job's real waiting time belongs. We preferred to address the problem as a classification task due to the bad results obtained through regression-based strategies (see Table 4). As reported in Section 4, the solution, compared with other experiments, shows promising results, although some classes' performance needs to be improved. The rather poor results associated with these classes are related to the unbalanced available dataset, considering that, as it can be seen from the graph in Figure 4, there is an uneven distribution between classes. In fact, the records used to predict the last five classes are about a third compared to those used to predict only the first one. As classification levels increase, the number of analysed jobs decreases while the variability of the pending time increase, therefore the available data is not sufficient to accurately model all classes. The decrease of available jobs for the training process is affected also by the misclassifications in the previous levels. Moreover, the variables used to model the multiple classification tasks did not perfectly model the problem under analysis; thus, the problem appears to be under-modeled. This means that the variables used to predict the categorical waiting-time for each job are not sufficient to describe the problem optimally (the high variability of different classes in the case of an unbalanced dataset). There is a lack of essential information for the considered algorithms to improve the system's classification performance. In this regard, it was noted that little data could describe the complexity of the job.
This work is a preliminary step towards implementing a scheduling system based on machine learning techniques on what we are currently working on. Based on the discussed experimental results, there is room for improvements in the proposed data-driven methodology. More specifically, we are currently working with HPC experts to define an extended set of attributes to model the problem addressed in this paper, understand how to get them and where, or create a new feature engineering step similar to the one proposed in [2] to improve the system's overall performance. Furthermore, we are working to extend the current methodology: (1) to predict other types of information (e.g., the number of CPUs necessary for a job, the termination status of the job execution) to enhance the HPC cluster performance and (2) to estimate the model degradation over time [4].

REFERENCES

[1] N. S. Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician* 46, 3 (Aug. 1992), 175–185. https://doi.org/10.1080/00031305.1992.10475879
[2] Daniele Apiletti, Claudia Barberis, Tania Cerquitelli, Alberto Macii, Enrico Macii, Massimo Poncino, and Francesco Ventura. 2018. iSTEP, an Integrated Self-Tuning Engine for Predictive Maintenance in Industry 4.0. In *IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom 2018, Melbourne, Australia, December 11-13, 2018*. IEEE, 924–931.
[3] Leo Breiman. 2017. *Classification and regression trees.* Routledge (Milton Park, Abingdon, Oxfordshire).
[4] Tania Cerquitelli, Stefano Proto, Francesco Ventura, Daniele Apiletti, and Elena Baralis. 2019. Towards a real-time unsupervised estimation of predictive model degradation. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE 2019, Los Angeles, CA, USA, August 26, 2019*. ACM, 5:1–5:6.
[5] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411
[6] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, and A. Miguel. 2016. A Practical Approach to Hard Disk Failure Prediction in Cloud Platforms: Big Data Model for Failure Management in Datacenters. In *2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService)*. 105–116. https://doi.org/10.1109/BigDataService.2016.10
[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning: data mining, inference and prediction* (2 ed.). Springer. http://www-stat.stanford.edu/~tibs/ElemStatLearn/
[8] Mordor Intelligence. 2020. High Performance Computing (HPC) Market, growth, trends, forecasts (2020 – 2025). Report posted

at. https://www.mordorintelligence.com/industry-reports/high-performance-computing-market

[9] M. Jassas and Q. H. Mahmoud. 2018. Failure Analysis and Characterization of Scheduling Jobs in Google Cluster Trace. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. 3102–3107. https://doi.org/10.1109/IECON.2018.8592822

[10] P. Li, B. Zhang, Y. Weng, and R. Rajagopal. 2017. A Sparse Linear Model and Significance Test for Individual Consumption Prediction. *IEEE Transactions on Power Systems* 32, 6 (2017), 4489–4500. https://doi.org/10.1109/TPWRS.2017.2679110

[11] Chunhong Liu, Liping Dai, Yi Lai, Guinbing Lai, and Wentao Mao. 2020. Failure prediction of tasks in the cloud at an earlier stage: a solution based on domain information mining. *Computing* 102 (2020), 2001–2023. https://doi.org/10.1007/s00607-020-00800-1

[12] C. Liu, J. Han, Y. Shang, C. Liu, B. Cheng, and J. Chen. 2017. Predicting of Job Failure in Compute Cloud Based on Online Extreme Learning Machine: A Comparative Study. *IEEE Access* 5 (2017), 9359–9368. https://doi.org/10.1109/ACCESS.2017.2706740

[13] Robert McGill, John W Tukey, and Wayne A Larsen. 1978. Variations of box plots. *The American Statistician* 32, 1 (1978), 12–16.

[14] J. M. Navarro, G. H. A. Parada, and J. C. Dueñas. 2014. System Failure Prediction through Rare-Events Elastic-Net Logistic Regression. In *2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation*. 120–125. https://doi.org/10.1109/AIMS.2014.19

[15] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy. 2006. Evaluation of a Workflow Scheduler Using Integrated Performance Modelling and Batch Queue Wait Time Prediction. In *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*. 29–29. https://doi.org/10.1109/SC.2006.29

[16] J. W. Park. 2019. Queue Waiting Time Prediction for Large-scale High-performance Computing System. In *2019 International Conference on High Performance Computing Simulation (HPCS)*. 850–855. https://doi.org/10.1109/HPCS48598.2019.9188119

[17] R. Polikar. 2006. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine* 6, 3 (2006), 21–45.

[18] Sebastian Raschka and Vahid Mirajalili. 2019. *Python machine learning*. Number 1. Packt Publishing,.

[19] A. Rosà, L. Y. Chen, and W. Binder. 2017. Failure Analysis and Prediction for Big-Data Systems. *IEEE Transactions on Services Computing* 10, 6 (2017), 984–998. https://doi.org/10.1109/TSC.2016.2543718

[20] S. S. SHAPIRO and M. B. WILK. 1965. An analysis of variance test for normality (complete samples)†. *Biometrika* 52, 3-4 (12 1965), 591–611. https://doi.org/10.1093/biomet/52.3-4.591 arXiv:https://academic.oup.com/biomet/article-pdf/52/3-4/591/962907/52-3-4-591.pdf

[21] Carlos N. Jr. Silla and Ale xA. Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery* 22, 1-2 (2011), 31–72. https://doi.org/10.1007/s10618-010-0175-9

[22] C. Spearman. 1904. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* 15, 1 (1904), 72–101.

[23] Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. 2018. *Introduction to Data Mining (2nd Edition)* (2nd ed.). Pearson.

[24] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. 2014. Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud. *IEEE Transactions on Cloud Computing* 2, 1 (2014), 14–28. https://doi.org/10.1109/TCC.2014.2306427