POLITECNICO DI TORINO
Repository ISTITUZIONALE

Predicting job execution time on a high-performance computing cluster using a hierarchical data-driven methodology

Original
Predicting job execution time on a high-performance computing cluster using a hierarchical data-driven methodology / Bethaz, P., Vacchetti, B., Capitelli, E., Nosenzo, V., Chiosso, L., Cerquitelli, T.. - (2022). (EDBT/ICDT Workshop, 6th International workshop on Data Analytics solutions for Real-LIfe APplications Edinburgh, UK 29th March-1st April, 2022).

Availability:
This version is available at: 11583/2961273 since: 2022-06-10T11:17:20Z

Publisher:
EDBT/ICDT

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

25 November 2026

Predicting job execution time on a high-performance Computing cluster using a hierarchical data-driven methodology

Paolo Bethaz[1], Bartolomeo Vacchetti[1], Enrica Capitelli[2], Vladi Nosenzo[2], Luca Chiosso[3] and Tania Cerquitelli[1]

[1]Department of Control and Computer Engineering, Politecnico di Torino, Italy
[2]Iveco Group, Torino, Italy
[3]NPO Torino Srl, Torino, Italy

Abstract

Nowadays, evaluating the performance of a vehicle before the production phase is challenging and important. In the automotive industry, many virtual simulations are needed to model the vehicle behavior in the best possible way. However, these simulations require a lot of time without the user knowing their runtime in advance. Knowing the required time in advance would allow the user to manage the simulations more effectively and choose the best strategy to use the available computational resources. For this reason, we present an innovative data-driven method to estimate in advance the execution time of simulations. Our approach integrates unsupervised techniques, such as constrained k-means clustering, with classification and regression algorithms based on tree structures.

In this paper, we present an innovative and hierarchical data-driven method for estimating the execution time of jobs. Numerous experiments were conducted on a real dataset to verify the effectiveness of the proposed approach. The experimental results show that the proposed method is promising.

Keywords

Execution-time prediction, data-driven model, hierarchical model

1. Introduction

Today, more and more manufacturing industries rely on either online data centers or physical HPC clusters to run a large variety of tasks to perform analyses and simulations. These tasks, or jobs, range from simulating individual mechanical components to analyze entire manufacturing processes. In this way, it is possible to shorten the time to market of the final product while reducing the number of errors made during the process. However, the execution of these jobs often requires resources that may not be immediately available, thus delaying the job execution and increasing the time needed to obtain the final results. In this paper, we present a data-driven methodology for predicting the jobs' execution time. We focus on predicting the execution time of simulations and analysis because it directly affects the waiting time of other jobs before they are submitted and the problem of unknown waiting time can lead to wasted cluster resources.

Since the number of analysis and simulations that must be performed for a single product is considerable, it is important to find a method to avoid wasting cluster resources and increasing delays. This issue is relevant in the context of software application development for the industrial domain and we want to address it by relying on an innovative methodology based on data analysis and machine learning techniques. In order to predict the execution time of jobs, we have taken into account not only the HPC resources required by the different jobs, but also the settings of the different solvers, that are the various kind of software used for analysis and simulation. Each simulation is characterized by a series of parameters (specific for each solver) that describe its configuration. These parameters are inserted manually from the user in phase of submission of the job and are then extracted in automatic way from the server used for running the simulations. The proposed approach is based on a hierarchical classification model. Our methodology is based on three separated models. The first model does a preliminary binary classification and then it divides the data accordingly. The other two models classify the two different portions of data, one portion of data for each model. In this way we are able to classify the data into four different classes while reducing the complexity of the task from a multiclass problem to a binary one at each step.

Published in the Workshop Proceedings of the EDBT/ICDT 2022 Joint Conference (March 29-April 1, 2022), Edinburgh, UK
paolo.bethaz@polito.it (P. Bethaz);
bartolomeo.vacchetti@polito.it (B. Vacchetti);
enrica.capitelli@external.cnhind.com (E. Capitelli);
vladi.nosenzo@ivecogroup.com (V. Nosenzo);
luca.chiosso@external.nposervices.com (L. Chiosso);
tania.cerquitelli@polito.it (T. Cerquitelli)
0000-0001-5016-8635 (P. Bethaz); 0000-0001-5583-4692 (B. Vacchetti); 0000-0002-9039-6226 (T. Cerquitelli)
© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

The rest of the paper is divided as follows. Section 2 deals with the literature review related to HPC, from resource allocation to runtime prediction. Section 3 deals with the proposed methodology, while Section 4 presents our results so far. Finally, in Section 5, we discuss our methods and future steps to continue this research.

2. Literature Review

There are several approaches and studies investigating how to improve HPC resource allocation. Research activities can be classified as: (i) predicting job failures, (ii) developing scheduling algorithms based on machine learning techniques, and (iii) using simulation execution time estimation to predict the waiting time of jobs that have yet to be submitted.
The first research strand's goal is to estimate if a specific job will fail or complete its execution. The intent is to stop prematurely those jobs that are predicted by the algorithms as failures [1, 2, 3]. By learning which jobs are more likely to fail and stopping them the HPC is able to improve its performance, while saving energy that would be wasted[4, 5]. However this type of approach requires a lot of data in order to identify a pattern between the attributes and the target variables. The most significant variables can be extracted in different ways, either through some feature engineering process or from different databases. Even so, parsing and transformation operations have been proven very useful in order to improve the prediction results [6] are extremely useful to obtain better prediction results. The issue here is related to the fact that some failures are rare, hence they are not easy to predict, but still consume a lot of resources [7]. For example, Liu et al. [1] integrated two algorithms in order to estimate whether a job fails or not. The first algorithm is a clustering one. It measures the correlation among jobs from different contexts. The other one is a multitask learning algorithm trained on correlated jobs. Since our data is not enough to achieve meaningful results in this paper we do not tackle this problem.
Another research approach focuses on the optimization of the available resources performed by a scheduler. After analyzing the behavior of successful and failed jobs, Jassas et al. [2] investigated scheduling algorithms intended to optimize the reliability and availability of cloud applications. Since the number of resources cannot be unlimited, large-scale HPC exploit waiting queues. However it has been proved by Nurmi et al. [8] that regardless of the performance of a resources scheduler one of the main factor that impacts the prediction efficiency is the amount of time that a job has to wait before being submitted. Following this idea, also authors in [9] try to predict the waiting time of a job using a hierarchical classification approach. However, the estimation of the waiting time is impacted by many factors, such as HPC specifics. Technical specifics aside, one factor that impacts the waiting time of every simulation is the execution time of the job that are running on the HPC. Some studies have focused on this approach, such as [10, 11]. While we agree on the centrality of the execution time we have adopted a different strategy compared to the studies mentioned before. As a matter of fact we use pretty much the same toolbox, i.e. clustering for data preprocessing and classification and or regression to estimate the execution time, however, as far as we know, we differentiate ourselves from previous work through the use of a hierarchical approach, which will be explained in detail in the following section.

3. Data-driven methodology

The building blocks of the proposed methodology, shown in Figure 1, are as follows: i) data cleaning, ii) model building and iii) model evaluation. Each of these steps is adequately described in the related subsection.

After the data collection phase, the generated dataset contains a record for each submitted job. These jobs may have been executed by different solvers. Here with solver we mean the software used to run the simulation, each of which is characterized by different model variables. Due to the different parameters that characterize each solver, and due to very different execution times between solvers, we decided to consider only one solver at a time, thus avoiding working with a dataset too sparse.

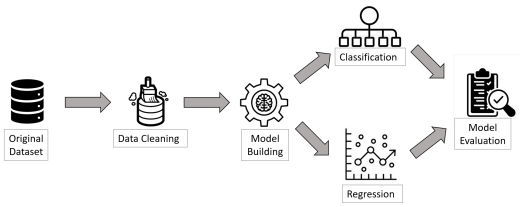


Figure 1: Schema of the proposed methodology

3.1. Data cleaning

Since the parameters characterizing each job are entered manually by the user during the job submission phase, it is essential to check the correctness of the available data before using them for subsequent analysis.
The data cleaning phase started with a collaboration with domain experts, who, thanks to their knowledge, helped us to define the admissibility ranges for each collected variable, including the execution time. This phase helped us to better understand the available data, and led us

to the decision of eliminating all those jobs associated with anomalous values of execution time. Specifically, all jobs with an execution time that was too low or too high compared to the execution time of other jobs run by the same solver were eliminated. To define when an execution time should be considered too high or too low, we tried using the following three approaches to define appropriate thresholds, beyond which a point must be considered an outlier [12]:
Interquartile Range (IQR): IQR is the difference between the values ranking in 25% (Q1) and 75% (Q3) in a data set. IQR thresholding strategy calculates the threshold as follows:

- min threshold: Q1 - (1.5*IQR)
- max threshold: Q3 + (1.5*IQR)

95th centile: the minimum threshold is set by taking the value ranking in 5% and the maximum threshold is set by taking the value ranking in 95%;
99th centile: the strategy is identical to that described in the previous point, but here the thresholds are defined so that fewer outliers are identified. The minimum threshold is set by taking the value ranking in 1% and the maximum threshold is set by taking the value ranking in 99%.
We compared the number of jobs labeled as outliers with each of the 3 approaches, and in subsequent experiments we tried to evaluate how the performance of a predictive model varies depending on the preprocessing used.

3.2. Model Building

The task of our model is to predict the execution time of a simulation running on a HPC. Since the goal is to predict a time that is a continuous value, this could be tagged as a regression task. However, the experimental results obtained by treating this task as a regression one led to poor results. This behavior can be justified by the fact that the data collection phase is quite recent, so the available data at the moment are not numerous. For this reason, we decided to treat it like a classification problem; this was made possible by categorizing the available runtimes, dividing them into classes representing contiguous time intervals. After this categorization, a classification algorithm can then try to predict in which range of values the execution time of the analyzed job will fall. In other words we have a multiclass problem. However, due to the scarce amount of data in our possession, the performance of our initial model was not enough. Since the amount of data is limited it is difficult for a classification algorithm to make good predictions in a multiclass context. On the other hand we did not want to oversimplify the problem by reducing the number of classes considered. Thus by implementing a classification hierarchical approach we were able to improve the goodness of the predictions without sacrificing too much quality. The good results obtained with the hierarchical classification approach pushed us to also try a mix between classification and regression algorithms. However the results, while in some cases were better than the normal regression approach, were not satisfying. This led us to choose the hierarchical classification approach. The following subsections describe in more detail which the structures, the algorithms and the techniques that were used, for both regression and classification approaches. Due to the fact that the amount of data in our possession is limited we have decided to rely on the XGBoost method with both the mixed regression and the classification approach. The XGBoost is a tree model that relies on the Extreme Gradient Boosting technique [13].

3.2.1. Classification Approach

Unlike a single level classification where the model is trained only once on all available data, in the hierarchical approach a binary tree structure is used, in which each node of the tree corresponds to a binary classification where a model predicts to which of the two classes the job belongs. The depth of the tree depends on the number of total classes that we want to obtain (each of which represents a temporal range of values). The hierarchical binary structure we used in this methodology has two levels of depth, to which correspond 3 predictive models (nodes) and 4 total classes (leaves). Solutions with different depths have also been tested experimentally, but the one with four classes has demonstrated to be the best compromise between good models' performance and enough detailed classes. An example of how the structure looks is reported in Figure 2. In this way every classifier has to deal with a binary classification problem, but overall the classes considered are four.

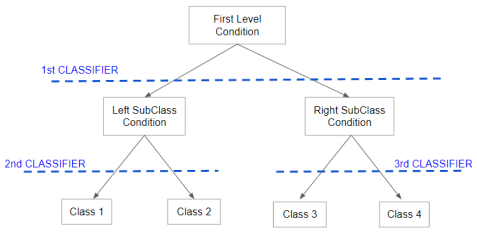


Figure 2: Hierarchical Structure Schema

From the figure, it is evident that the key of the entire structure are the three conditions that determine in which sub-branch the obtained prediction must go. To each of these conditions corresponds a subdivision of the dataset (or of a portion of it), in two classes that represent different time intervals of the execution time of the jobs. The overall performance of the classification method is

greatly influenced by the identified thresholds, so it is important to try to define them as best as possible. To do this, two different approaches have been tested:

- *balanced approach*: in each division of the identified hierarchical structure, the available data is divided into two classes, each containing the same number of jobs. This technique prevents any unbalanced class problem, allowing the predictive model to be trained on balanced classes;
- *k-means approach*: the classes to use for training a predictive model in each node of the structure, are chosen in an automatic way through a clustering algorithm. In particular, the k-means algorithm is used, with K=2. In addition, to prevent the proposed solution from leading to an unbalanced-classes problem, we used a constrained version of the k-means algorithm, in which a minimum size for each cluster can be specified. In particular, the constraint we used here is that each identified class had to contain at least 40% of the total jobs.

3.2.2. Mixed Approach

Several regression algorithms (XGBoost [13], RandomForest [14], Lasso [15]) were tested and compared with each other, evaluating their performance based on the R2 value obtained. The Lasso Regression is a regression analysis method that enhance its prediction accuracy by combining variable selection and regularization techniques; while RandomForest and XGBoost are both tree-based algorithms exploiting several decision trees, differing from each other on how the trees are constructed and how the results are combined.
Using a regression algorithm has the advantage of yielding a punctual value of the estimated runtime. However, due to limited availability of initial data, the obtained predictive model built on a single level is not very robust. For this reason, we have decided to test a mixed hierarchical regression approach. With mixed approach we mean that there is a combination between classification and regression. At the first prediction layer we have a classifier, while at the second prediction layer we have used two regressors. Thus the two regressors at the second prediction layer have to estimate the execution time of jobs that belong to two different time intervals, one for each model. In this way we simplified the problem while keeping the final prediction as a continuous value. Figure 3 shows the scheme of the proposed mixed approach. Regarding the techniques used to obtain the two classes in the first level of classification, both approaches described in the previous classification case were tested.

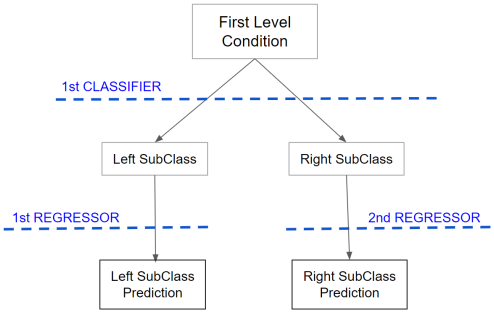


Figure 3: Mixed Approach Structure

3.3. Model Evaluation

For the evaluation of every algorithm used in all the performed experiments, we exploited the Leave-One-Out Cross Validation (LOOCV) technique. Even if it is a computationally expensive technique, LOOCV results in a reliable and unbiased estimate of model performance. Moreover, this method turns out particularly useful when the data available are limited, like in our case in which the phase of data collection is begun recently.
LOOCV is an extremization of k-fold validation, where the value of k is equal to the number of available items (N). Its operations can be summarized in the following steps:

1. Split the dataset into N disjoined groups, where each group contains a single element;
2. For each group:
   - Take the element in the considered group as test set
   - Take the remaining N-1 groups as training set
   - Fit a model on the training set and evaluate it on the test set
   - Retain the evaluation of the model and then discard it
3. The model performance is estimate as the average of the N experiments executed

4. Preliminary experimental results

The experiments presented here show the actual results obtained and that motivate us to rely on the hierarchical classification approach. Section 4.1 offers insight on the data that we have used to train our models. Section 4.2 discusses the effectiveness of the proposed techniques to correctly identify outliers and how they impact the selectivity of the dataset cardinality. Section 4.3 shows

the performance of classifier models, both normal and with the hierarchical structure. Section 4.4 presents the results obtained with different regression algorithms and the hierarchical mixed approach.

4.1. Dataset Description

Our data was extracted from a PBS (Portable Batch System) server used to run simulations of various nature, from aerodynamics to virtual crash tests, related to the automotive context. Our data belong to two main categories, i.e. explicit and implicit jobs. The implicit methods use an algorithm "step by step", in which an appropriate convergence criterion allows the analysis to continue or not, reducing the time increment, depending on the accuracy of the results at the end of each step. Using the explicit methods does not have problems of non-convergence, since in this case the time increment is defined at the beginning and remains constant during the calculation. After the data collection phase, the dataset contains about 6000 records, each of which represents a job submitted in the cluster. Jobs can be performed by five different solvers, depending on the type of analysis to be run. A summary of the solvers contained in our dataset is given in Table 1. Due to the different parameters that characterize each solver, and due to very different execution times between solvers, the analyses described below consider only one solver at a time.

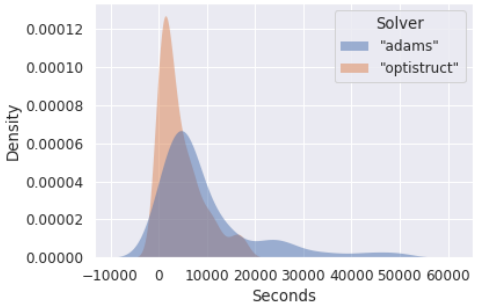


Figure 4: KDE plot for execution times of two different solvers

As a demonstration of the differences between the solvers, Figure 4 shows the kernel density estimate (KDE) plot for the execution times of an explicit solver (Adams) and an implicit solver (Optistruct). KDE represents the data using a continuous probability density curve and the x-axis in the figure show how the jobs belonging to the two solvers occupy very different ranges of execution times, with much greater times for the explicit solver.

Table 1
Solvers

Type	Solver name	Application Field
Explicit	Adams	Multi body approach
Explicit	Radioss	Crash Simulation
Implicit	Abaqus	Linear/Non linear analysis
Implicit	Nastran	FE Analysis
Implicit	Optistruct	FE Analysis and optimization

4.2. Data Cleaning

In this preprocessing phase we tried to remove all the jobs having an anomalous execution time compared to the runtimes of the other jobs. To do this, 3 different methodologies were tested as discussed in Section 3.1, based on: i) interquartile range (IQR), 95th centile, 99th centile. The percentage of jobs labeled as outliers by each of the three techniques, separately by solver, is shown in Table 2.

Table 2
The number of outlier jobs for each solver

Solver	IQR	95th centile	99th centile
Adams	14%	10%	2%
Radioss	1%	10%	2%
Nastran	11%	10%	3%
Optistruct	8%	11%	3%
Abaqus	11%	10%	2%

Since we can not know in advance which of the 3 techniques will lead to greater benefits, in the following analysis we compared the results obtained with different preprocessing techniques, evaluating then the best of them. However, from Table 2 we can see that the IQR and 95th centile techniques show rather similar results (with an average difference of about 3%); instead the 99th centile technique often identifies a very low percentage of outliers compared to the other techniques. For this reason, in the following analyses we will consider only the IQR and the 99th centile techniques, comparing the results obtained with these two different approaches.

4.3. Classification Model Evaluation

We have conducted a series of experiments with the proposed hierarchical approach, testing different preprocessing techniques and different strategies for identifying thresholds.

Table 3 contains the F-score values obtained using the 99th centile as preprocessing step and the k-means for thresholds identification, since it is the configuration with which the best results were obtained. For each solver, the first column of the table shows the results

Table 3
F-score for Hierarchical Classification

	1st level classification		2nd level classification			
Solver	**1**	**2**	**1**	**2**	**3**	**4**
Adams	0.90	0.82	0.91	0.86	0.82	0.66
Radioss	0.71	0.72	0.76	0.25	0.79	0.61
Nastran	0.85	0.79	0.89	0.86	0.93	0.90
Optistruct	0.91	0.94	0.89	0.81	0.82	0.64
Abaqus	0.82	0.69	0.81	0.67	0.88	0.80

for the first classification level (where the first two classes are identified), while the second column contains the F-score values obtained in the second level of classification (classes 1 and 2 for the left sub-branch, classes 3 and 4 for the right sub-branch). To better illustrate our methodology, a focus on a specific solver is also represented in Figure 5, that shows the hierarchical approach specifically for the *Nastran* solver, indicating the relevance of the identified classes in a real context. The predictive model used in all the nodes is the XGBoost and on the figure are indicated the F-Score values for each prediction. Here, the implemented model is able to predict quite efficiently whether the execution time of an analyzed job will be less than 8 minutes, between 8 and 16 minutes, between 16 and 30 minutes, or greater than 30 minutes. For solvers with different characteristics obviously different thresholds will be obtained; however the results in the Table 3 indicate that, except for the Radioss solver (where the results obtained are below the average behavior), for all solvers we are able to predict quite accurately which class a job should belong to.

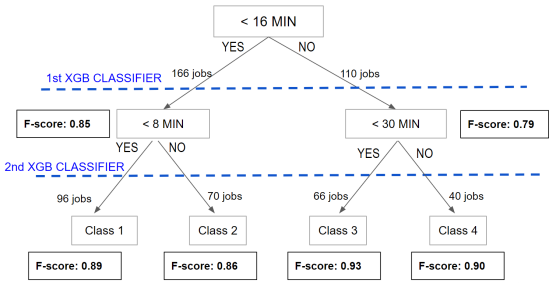


Figure 5: Nastran Hierarchical Approach

In order to validate the results obtained with the hierarchical approach, we conducted also a series of experiments with a more traditional methodology and we compared them with our results. The traditional methodology consists in a "single" level approach, in which the algorithm used is always an XGBoost, but there is no a hierarchical structure. The results in Table 4 represent the baseline against which we want to compare our methodology. The table shows the results obtained with both the two preprocessing techniques (IQR vs 99th centile) and with both the strategies to define the subdivision in classes (k-means vs balanced approach). For reasons of space, each solver has been indicated in this table only by its initials (Ad, R, N, O, Ab); moreover, 'K-M' stands for k-means, while 'Bal' indicates the balanced approach. Comparing the f-score values obtained in the two methodology, it is evident how the hierarchical structure allows to obtain better performances than the baseline approach, whatever preprocessing technique is used.

Table 4
Baseline Classification F-score

	IQR				99%			
Solver	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
Ad. K-M	0.79	0.52	0.28	0.75	0.80	0.63	0.46	0.60
Ad. Bal	0.75	0.64	0.49	0.72	0.78	0.71	0.59	0.71
R. K-M	0.44	0.44	0.23	0.14	0.44	0.48	0.23	0.62
R. Bal	0.26	0.55	0.54	0.57	0.21	0.35	0.42	0.64
N. K-M	0.66	0.49	0.87	0.57	0.85	0.75	0.66	0.57
N. Bal	0.83	0.83	0.73	0.67	0.79	0.69	0.71	0.65
O. K-M	0.67	0.84	0.68	0.68	0.67	0.64	0.73	0.84
O. Bal	0.64	0.60	0.56	0.76	0.72	0.54	0.70	0.77
Ab. K-M	0.56	0.77	0.58	0.55	0.76	0.45	0.65	0.75
Ab. Bal	0.60	0.61	0.57	0.62	0.55	0.69	0.60	0.64

4.4. Mixed Model Evaluation

In the approach that exploit regression algorithms to estimate the execution time, the output of the predictive model is a continuous numerical value that indicates the runtime of the considered jobs. Table 5 contains the R2 values obtained with three different Regressors: RandomForest Regressor (RF), XGBoost Regressor and Lasso Regressor. The results obtained with RandomForest and XGBoost are very similar, both of them much better than the results obtained with a Lasso Regressor, which does not perform well. Furthermore, from the baseline table, we see that the results obtained after removing the outliers through the 99th centile, are on average higher than those obtained using the IQR. For this reason we use the 99th centile technique for testing our approach. Currently, the mixed approach is still an experimental approach and for now it has been tested on only one solver. The results are shown in Figure 6, where the considered solver is Nastran and the algorithm used is the XGBoost (both for classification and regression). We can see that the first classification level is the same obtained in Figure 5, with the same values of F1-score. Then, unlike the classification approach, in the mixed approach we used two regression models in the second level (one for each sub-branch), able to predict the value of execution time of the considered job.

By having a first classification level that can distin-

Table 5
Baseline Regression R2

	RF		XGBoost		Lasso	
Solver	**IQR**	**99%**	**IQR**	**99%**	**IQR**	**99%**
Adams	0.31	0.70	0.36	0.69	0.23	0.27
Radioss	0.27	0.27	0.27	0.28	0.09	0.12
Nastran	0.38	0.39	0.40	0.37	0.08	0.09
Optistruct	0.63	0.45	0.60	0.44	0.40	0.43
Abaqus	0.36	0.41	0.38	0.42	0.10	0.09

guish two categories of jobs, the R2 values obtained in the second regression level are now higher than those obtained with the Nastran solver using the baseline approach. Moreover, the mean absolute error (MAE) values shown in Figure 6, indicates that the average error associated with jobs that last less than 16 minutes is just over one minute (69 seconds), therefore an acceptable error in our use-case. Regarding instead the mean absolute error associated with jobs that last longer than 16 minutes, this one turns out to be about 11 minutes, so a bit more impactful.

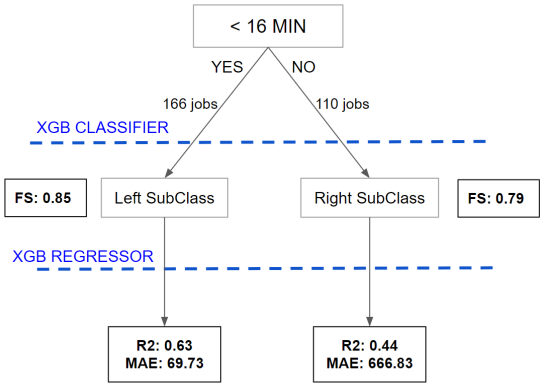


Figure 6: Nastran Mixed Approach

The error grows as execution times increase. So, for the considered solver, the better choice could be to adopt techniques of regression in order to estimate low execution times; and to use instead a classification approach in order to predict the classes to which the job will belong when it deals with longer execution times.

5. Discussion and Future Research direction

We have presented a classification hierarchical model which is able to address a multiclass problem by dividing its complexity among multiple prediction layers. In this way every classifier has to deal with a binary classification problem, instead of a multiple classification. Even if the amount of data in the considered use case is limited our approach has shown a promising performance also compared to single prediction level approaches. Especially in the classification context the hierarchical approach performs better than the normal approach. We have also tried some experiments in which more than one solver is taken into account. However, due to the fact that every solver takes into account a different set of variables the resulting dataset is very sparse. This issue combined with the limited amount of data leads to poor predictions with both the hierarchical approach and a single prediction level method. Once that the data in our possession has reached a higher numerosity, it will allow us to investigate whether or not our hierarchical approach can outperform more classical approaches in a more complex environment. A higher amount of data means that the impact of the different variables taken into account will be reduced. Currently we are still working on this project and we already have different improvements that we want to address. We will keep gathering more data that will allow us to build more stable and robust models. We also intend to further investigate the possibility of building a model that is able to make predictions on the whole set of solvers, instead of relying on a different model for every solver. Finally we intend to integrate our model inside the HPC structure in order to help it assess the pending time of newly submitted jobs.

References

[1] C. Liu, L. Dai, Y. Lai, G. Lai, W. Mao, Failure prediction of tasks in the cloud at an earlier stage: a solution based on domain information mining, Computing 102 (2020) 2001–2023. doi:https://doi.org/10.1007/s00607-020-00800-1.

[2] M. Jassas, Q. H. Mahmoud, Failure analysis and characterization of scheduling jobs in google cluster trace, in: IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Omni Shoreham, United States, 2018, pp. 3102–3107. doi:https://doi.org/10.1109/IECON.2018.8592822.

[3] C. Liu, J. Han, Y. Shang, C. Liu, B. Cheng, J. Chen, Predicting of job failure in compute cloud based on online extreme learning machine: A comparative study, IEEE Access 5 (2017) 9359–9368. doi:https://doi.org/10.1109/ACCESS.2017.2706740.

[4] A. Rosà, L. Y. Chen, W. Binder, Failure analysis and prediction for big-data systems, IEEE Transactions on Services Computing 10 (2017)

984–998. doi:https://doi.org/10.1109/TSC.2016.2543718.

[5] P. Li, B. Zhang, Y. Weng, R. Rajagopal, A sparse linear model and significance test for individual consumption prediction, IEEE Transactions on Power Systems 32 (2017) 4489–4500. doi:https://doi.org/10.1109/TPWRS.2017.2679110.

[6] S. Ganguly, A. Consul, A. Khan, B. Bussone, J. Richards, A. Miguel, 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), 2016 IEEE Second International Conference on Big Data Computing Service and Applications, Oxford, United Kingdom, 2016, pp. 105–116. doi:https://doi.org/10.1109/BigDataService.2016.10.

[7] J. M. Navarro, G. H. A. Parada, J. C. Dueñas, System failure prediction through rare-events elastic-net logistic regression, in: 2014 2nd International Conference on Artificial Intelligence, Modelling and Simulation, IEEE, Madrid, Spain, 2014, pp. 120–125. doi:https://doi.org/10.1109/AIMS.2014.19.

[8] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, K. Kennedy, 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, IEEE, Tampa, FL, USA, 2006, pp. 29–29. doi:10.1109/SC.2006.29.

[9] F. Carfi, E. Capitelli, V. M. Nosenzo, T. Cerquitelli, Estimating the job's pending time on a high-performance computing cluster through a hierarchical data-driven methodology, in: DOLAP, EDBT 2021, Nicosia, Cyprus, 2021.

[10] M. Ferro, V. P. Klôh, M. Gritz, V. de Sá, B. Schulze, Predicting runtime in hpc environments for an efficient use of computational resources, in: Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho, SBC, WSCAD 2021 – XXII Simpósio em Sistemas Computacionais de Alto Desempenho, Belo Horizonte, 2021, pp. 72–83.

[11] H. Wang, Y.-Q. Dai, J. Yu, Y. Dong, Predicting running time of aerodynamic jobs in hpc system by combining supervised and unsupervised learning method, Advances in Aerodynamics 3 (2021). doi:10.21203/rs.3.rs-360961/v1.

[12] J. Yang, S. Rahardja, P. Fränti, Outlier detection: How to threshold outlier scores?, in: Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing, AIIPCC '19, Association for Computing Machinery, New York, NY, USA, 2019. URL: https://doi.org/10.1145/3371425.3371427. doi:10.1145/3371425.3371427.

[13] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 785–794. URL: https://doi.org/10.1145/2939672.2939785. doi:10.1145/2939672.2939785.

[14] L. Breiman, Random forests, Machine learning 45 (2001) 5–32.

[15] R. Tibshirani, Regression shrinkage selection via the lasso, Journal of the Royal Statistical Society Series B 73 (2011) 273–282. doi:10.2307/41262671.