Learning-based Approaches to Estimate Job Wait Time in HTC Datacenters

Luc Gombert and Frédéric Suter

IN2P3 Computing Center / CNRS, Lyon-Villeurbanne, France
firstname.lastname@cc.in2p3.fr

Abstract. High Throughput Computing datacenters are a cornerstone of scientific discoveries in the fields of High Energy Physics and Astroparticles Physics. These datacenters provide thousands of users from dozens of scientific collaborations with tens of thousands computing cores and Petabytes of storage.
The scheduling algorithm used in such datacenters to handle the millions of (mostly single-core) jobs submitted every month ensures a *fair sharing* of the computing resources among user groups, but may also cause unpredictably long job wait times for some users. The time a job will wait can be caused by many entangled factors and configuration parameters and is thus very hard to predict. Moreover, batch systems implementing a fair-share scheduling algorithm cannot provide users with any estimation of the job wait time at submission time.
Therefore, we investigate in this paper how learning-based techniques applied to the logs of the batch scheduling system of a large HTC datacenter can be used to get an estimation of job wait time. First, we illustrate the need for users for such an estimation. Then, we identify some intuitive causes of this wait time from the information found in the batch system logs. We also formally analyze the correlation between job and system features and job wait time. Finally, we study several Machine Learning algorithms to implement learning-based estimators of both job wait time and job wait time ranges. Our experimental results show that a regression-based estimator can predict job wait time with a median absolute percentage error of about 54%, while a classifier that combines regression and classification assigns nearly 77% of the jobs in the right wait time range or in an immediately adjacent one.

1 Introduction

High Energy Physics and Astroparticles Physics experiments are heavy consumers of computing resources. Numerical simulations of physical processes generate massive amounts of data that are compared to data produced by detectors, satellites, or telescopes. The analysis and comparison of these experimental and simulated data allow physicists to validate or disprove theories and led to major scientific discoveries over the last decade. In 2012, two experiments running on the Large Hadron Collider (LHC) at CERN, both observed a new particle which is consistent with the Higgs boson predicted by the Standard Model. In 2016,

the LIGO and VIRGO scientific collaborations announced the first observation of gravitational waves which confirmed the last remaining unproven prediction of general relativity. In both cases, these observations were awarded a Nobel Prize.

A characteristic shared by many physics experiments is that their computing models rely on single-core but numerous, and sometimes very long lasting, jobs, e.g., Monte-Carlo simulations and data analyses, to obtain scientific results. Then, this scientific community benefits more of High Throughput Computing (HTC) than High Performance Computing (HPC).

The Computing Center of the National Institute of Nuclear Physics and Particle Physics (CC-IN2P3) [21] is one of the thirteen Tier-1 centers in the *Worldwide LHC Computing Grid* (WLCG) engaged in the primary processing of the data produced by the LHC. About 2,500 users from more than 80 scientific collaborations share nearly 35,000 cores to execute a large HTC workload of about 3 million jobs per month. These resources are managed by Univa Grid Engine [23] which implements the *Fair Share Scheduler* [10] and thus assigns priorities to all the unscheduled jobs to determine their order of execution. HTC jobs being in a vast majority single-core jobs, scheduling is much easier than with parallel jobs in HPC systems. The main operational objectives are to maximize resource utilization and ensure that every group is served according to its expressed resource request for the year.

In a previous study we showed that two distinct sub-workloads are executed at CC-IN2P3 [2]. Some jobs are submitted by a small number of large user groups through a *Grid* middleware, at a nearly constant rate and with an important upstream control of the submissions while *Local* users from about 60 different groups directly submit their jobs to the batch system. We also showed that the jobs submitted by Local users suffer from larger wait times than Grid jobs. Job wait time can even become unpredictably long for some users and is caused by many entangled factors and configuration parameters. It is thus very hard to predict and may lead to a poor Quality of Service. Moreover, the Fair-Share scheduling algorithm cannot provide users with any estimation of job wait time at submission time as other scheduling algorithms, e.g., Conservative Backfilling [15], can do.

In this work we investigate how learning-based techniques applied to the logs of the batch scheduling system of a large HTC datacenter can be used to provide users with an estimation of the time their jobs will wait when they submit them. While this study focuses on the specific configuration and workload of the CC-IN2P3, we believe that our findings can be straightforwardly applied to other large HTC datacenters involved in the WLCG that show common characteristics. To this end, we make the following contributions:

- Motivate the need for a job wait time estimator.
- Identify some intuitive causes of the job wait time.
- Formally analyze the correlation between job and system features and job wait time.
- Propose a learning-based estimator of job wait time and a classifier in wait time ranges.

The remaining of this paper is organized as follows. Section 2 presents the related work. We analyze in Sect. 3 the distribution of the job wait time and detail some of its intuitive causes. In Sect. 4 we confront these intuitive causes to the correlation of job wait time with job and system features. Section 5 details the proposed learning-based approaches and presents our experimental results. We discuss the applicability of the proposed work to other workloads in Sect. 6. Finally, Section 7 summarizes our findings and outlines future work directions.

2 Related Work

Knowing when a job will start when it is submitted, and thus for how long it will wait in a queue, is a long-time concern, and a well-studied problem, for batch-managed datacenters. This is even more important when users have access to more than one datacenter. Then, job wait time becomes an important component of a *meta-scheduling* process to decide where to submit a job. The Karnak service [18] was for instance deployed on TeraGrid to predict job wait time within a certain confidence interval for the different sites composing the infrastructure, before or once a job is submitted. Karnak maintains a database of job features, system state, and experienced wait time and then derives a prediction for a new job by finding similar entries in this database. This approach is referred as *Instance Based Learning* [12,13]. Another technique is to predict job wait time as a range (e.g., between 1 and 3 hours), for instance by using a k-Nearest Neighbors algorithm to select similar instances and then refine the prediction using Support Vector Machines [11]. In addition to these similarity-based approaches, some works directly use job and/or system features to predict upper bounds [4], by matching distributions, or ranges [9], using a Naive Bayes classifier, for job wait time. Another approach consists in defining job templates, leveraging the similarity of a new job with historical information to estimate its runtime, and then simulate the behavior of the batch scheduling algorithm to derive the wait time of each job [19,20]. However, such a simulation-based approach is too compute-intensive to build an online estimator.

In this work, we apply Machine Learning (ML) techniques to both job and system features to determine into which of the predefined time ranges the wait time of a new job will fall. Moreover, all the aforementioned works consider HPC workloads where the size of a job, in terms of both number of cores/nodes and requested runtime, play an important role in job wait time. Indeed, the more cores a job requests, the harder it is to fit in a schedule, hence the longer it may wait. Moreover, long jobs are bad candidates for backfill. Here, we consider a HTC workload made of a vast majority of single-core jobs scheduled with no backfill. Then, the causes of job wait time are completely different in our case.

3 Wait Time Distribution and Intuitive Causes

To understand the causes of job wait time, we analyze the workload executed on the resources of the CC-IN2P3 over 23 weeks from Jun. 25, 2018 to Dec. 2,

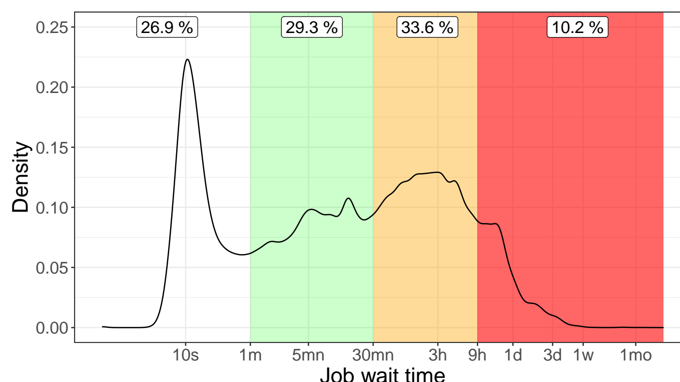

Fig. 1. Probability Density Function of Local job wait time.

2018. This corresponds to a stable period during which nearly 35,000 cores were made available to the users. This workload is composed of 7,749,500 *Grid* jobs and 5,748,922 *Local* jobs, for a total of 13,498,422 jobs. Hereafter we focus only on the Local jobs, as they experience larger wait times than Grid jobs [2].

Figure 1 shows the Probability Density Function (PDF) of Local job wait time. For the sake of readability, we used a logarithmic scale on the x-axis and highlighted four regions. The labels at the top of the graph show the respective percentage of jobs in each region.

The leftmost region corresponds to jobs that start almost right after their submission, i.e., within a minute. It represents more than one fourth of the total workload. For these jobs, and the users who submitted them, the Quality of Service is very good. Jobs in the second to left region experience a wait time between 1 and 30 minutes, which can be seen as reasonable. Indeed, users are asked to submit jobs whose requested runtime is at least of one hour. These two regions combined amount for 56.2% of the submitted Local jobs.

What really motivates this work are the remaining two regions. We can see on Fig. 1 that the job wait time of about one third of the workload spans from 30 minutes to 9 hours following a lognormal distribution. Then, it is hardly possible for users whose jobs fall into this category to guess when they will start. However, having an estimation of this delay may have a direct impact on their working behavior [17]. If a user knows that their job will start within the next hour, and also knows for how long this job will last, they may want to wait for the job completion before submitting other jobs. Conversely, if the user knows the job will not start before several hours, they can proceed with other activities and come back the next day to get the results. For the jobs in the last category, which amount for 10% of the workload, a wait time of more than nine hours corresponds to a poor Quality of Service experienced by the users.

The Quality of Service experienced by the users, i.e., the job *response time*, does not only depend on how much time the job waits but also on its execution

time. For instance, a wait time of two hours does not have the same impact whether the job lasts for one hour or one day. The *bounded slowdown* metric [5] captures this impact of job wait time on the response time. Using a bound on job execution time of 10 minutes, this metric indicates that 25% of the jobs in the third (resp. fourth) region wait between 3.2 and 8.5 (resp. 10.6 and 58.6) times their execution times. By further analyzing the distribution of job execution time, we observed that more than half of the jobs in these two regions run for less than 3 hours, and nearly 75% complete in less than 6 hours. Job wait time is thus not only unpredictable but can also be highly detrimental to users.

To understand which of the many different and entangled factors and configurations parameters can cause large job wait times, we propose to answer to the four following basic questions:

Who submits the job? Each entry in the logs of the batch system corresponds to a job and comprises two fields that allow us to identify which *user* from which *user group* is submitting a job: `owner` and `group`. According to the resource allocation policy of the CC-IN2P3 and the pledges made by the scientific collaborations for the year [1], each user group is allocated a *share* of the total available computing power proportional to its needs. This defines a consumption objective used by the job scheduler to compute its fair-share schedule.

What is the job requesting? A job is mainly characterized by the *time* the user estimates it will run and the *memory* it will need. These quantities are expressed at submission time by setting *hard* or *soft* limits through flags provided by the batch system. If a job hits a hard limit, it is killed and its results are lost. Using a soft limit allows the job to catch a signal before being killed, and thus to react accordingly. Time can be expressed as an expected runtime (i.e., `s_rt` and `h_rt` flags) or CPU time (i.e., `s_cpu` and `h_cpu` flags), both expressed in seconds. Memory can be expressed as a resident set size (i.e., `s_rss` and `h_rss` flags) or virtual memory size (i.e., `s_vmem` and `h_vmem` flags), both in bytes. If no value is given for these flags, default values depending on the submission queues are applied. Users can also specify the number of cores, or *slots* in the UGE terminology, and whether the job requires a specific resource, e.g., access to a given storage subsystem, database, or licensed software. To prevent the saturation of these critical resources or the violation of licenses, the batch administrators set up several limits as *Resource Quota Sets* (RQSs). These limits can be applied globally or on a per group basis and change over time.

When is the job submitted? The batch system logs submission times as timestamps from which the *hour* and *day* of submission can easily be derived.

Where is the job submitted? During the considered time period, jobs could be submitted to 6 different scheduling queues that mainly differ by maximum allowed duration, both in terms of runtime and CPU time, available memory and scratch disk space per job, and the type of jobs allowed to enter the queue, i.e., single- or multi-core. The bulk of the Local jobs is directed to the generic `long` queue while the others can accommodate jobs with special needs. This queue can access to almost all the available cores but has a rather low priority.

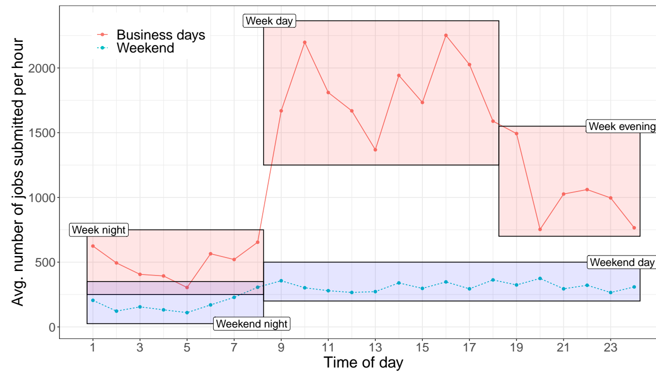

Fig. 2. Daily submission rate for Local jobs on business days or during the weekend.

By analyzing all this information we identify some *intuitive causes* of why a job would wait more than another that we detail hereafter. The first two identified causes come from when a job is submitted.

3.1 Submission Period

Fig. 2 shows the daily submission rate for Local jobs. We distinguish five different periods in this graph. First, there are much less submissions during the weekend than over the rest of the week, with a slight day/night difference. Then, on business days, e.g., Monday to Friday, we clearly see more submissions during the working hours than over night. We also distinguish a "*Week evening*" period whose submission rate is between those of the working hours and the night. Such a submission pattern is classical and representative of most HTC and HPC centers. As more concurrent job submissions obviously lead to more competition for resources, we can suppose that a job submitted during a week day is likely to wait more than a job submitted in around midnight or on a Sunday morning.

3.2 Number of Pending Jobs in Queue

The number of jobs already waiting in queues can also influence job wait time. To illustrate this, we focus in Fig. 3 on a typical 5-day period. The top graph shows the evolution of the total number of pending slot requests while the bottom graph displays how many jobs are waiting for a certain amount of time in each period. For the sake of readability, we sampled the logs using a 3-hour range, but a more detailed sampling confirmed our observations.

We can see two consequences of an increase of the number of pending jobs on job wait time. First, when a burst of submission occurs in the `long` queue, e.g., on Monday between 8 and 9 AM or on Wednesday around noon, we observe a dramatic increase of job wait time with many jobs waiting for more than twelve hours. Second, when a burst of submission occurs in another queue, e.g., on

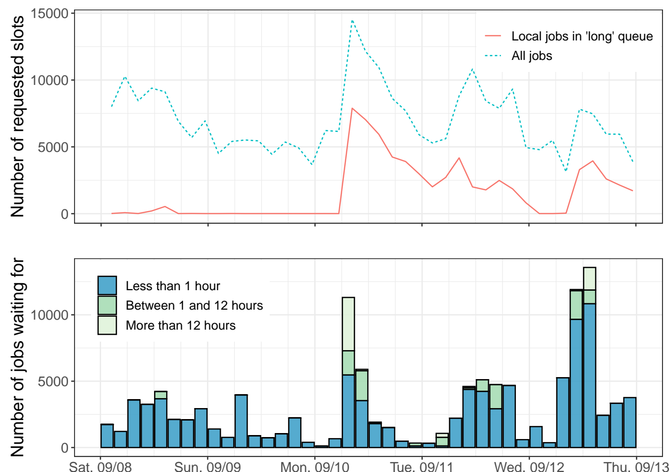

Fig. 3. Evolution of the number of requested slots by jobs waiting in queue (top). Distribution of wait time for local jobs in each 3-hour period (bottom).

Tuesday around noon, we also observe an increase of the number of jobs waiting between one and twelve hours. This can be explained by the fact that the burst happens in a queue of higher priority which causes delays for a large fraction of the jobs submitted to the less priority `long` queue. This focus outlines the importance of taking the *system state* into account at the queue and global levels when estimating job wait time.

3.3 Share Consumption

The concept of *shares* is at the core of the scheduling algorithm ran by UGE. First, it allows the batch system to give higher priorities to jobs submitted by user groups with bigger needs. With nearly 80 user groups to serve, whose size and needs are very heterogeneous, the allocated shares cover a wide range of values with differences up to two orders of magnitude between some groups. Then, a job submitted by a user from a small group with a small share is likely to wait more than a similar job from another larger group with a bigger share.

The scheduling algorithm also takes into account the recent resource consumption, over a sliding time window, of the different groups. If a user group starts to submit jobs after a period of inactivity, these jobs will get a priority boost. Conversely, a group that consumed a lot of resources over a short period of time will get some priority penalty. Finally, if a group consumed all its allocated share, its jobs will be executed if and only if there are still some resources available once jobs from the other groups are scheduled. Our job wait time estimator thus has to consider the initial share allocated to a group, which fraction of it has already been consumed, and the recent resource consumption of the group to estimate whether a job will be delayed or not.

3.4 Quotas on Resource Usage

A job can also see its wait time increase, sometimes dramatically, because of the different quotas, or RQSs, set by the administrators of the batch system. If a RQS is violated, at any level, jobs are blocked in queue until the quota violation is solved. We illustrate the impact of such quotas on job wait time in Fig. 4 with the extreme case of a single user who submitted more than 2,000 jobs in one minute on Nov. 14, 2018 at 6:20PM. The vertical line shows the submission burst while each black segment represents the execution of a single job.

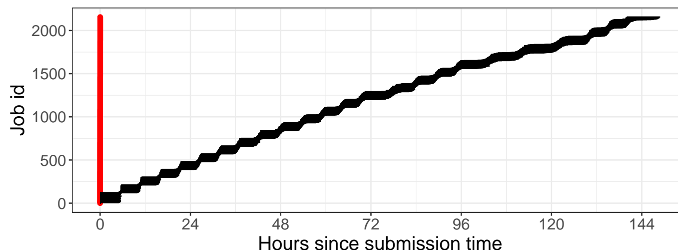

Fig. 4. Execution span of jobs submitted by a single user limited by a stringent RQS. The verticle line shows the submission burst and each black segment represents the execution of a job.

We can see that all the submitted jobs cannot start at once, but are executed by batches of 90 jobs, which corresponds to the limit applied to the group of this user on how many jobs requesting access to a specific storage subsystem can run concurrently. The direct consequence of this stringent quota (which was later increased and eventually removed) is that the last jobs wait for nearly six days before starting their execution. We can conclude from this example that quota violations must be taken into account by our estimator.

3.5 Resource Requests

Finally, the study of the related work presented in Section 2 showed that the resource requests made by a job, in terms of runtime, number of cores, or memory, can have a great impact on job wait time. However, this observation is valid for HPC systems where jobs can have different shapes and backfill mechanisms are implemented by the batch system. In the specific context of HTC datacenters such as CC-IN2P3, the vast majority of the jobs are single-core. Then, the produced schedule is almost free of idle times which makes backfilling meaningless. Moreover, the queue configuration neither reflects nor leverages the fact that 14% of the jobs express a runtime of less than 6 hours and 41% of less than 24 hours. Time and memory requests should nevertheless be taken into account but their impact is expected to be smaller than for traditional HPC workloads.

4 Job Features Correlation Analysis

In the previous section we identified a set of intuitive causes of job wait time. To confirm them, we extracted a set of job and system features related to these causes from the information available in the logs and conducted a detailed pair-wise correlation analysis of these features. This allowed us to merge the eight metrics related to the requested time and memory into only two features. We also replaced the day and hour of submission features by the five categories from Fig. 2 (i.e., week day, week evening, week night, weekend day, and weekend night). Finally, we made a Primary Component Analysis (PCA) on eleven features describing the system state. This analysis allows us to keep 99.98% of the observed variance with only four generated features. We verified that these modifications did not imply an important loss of information or accuracy. The obtained reduced set of features that directly derive of our analysis of the intuitive causes of job wait time is summarized in Table 1. We will use them to design the learning-based job wait time estimators detailed in the next section.

4.1 Spearman's Rank Correlation of Numerical Features

To determine the influence of each of our numerical job and systems features, we compute their respective correlation with job wait time. As the distribution of several of these features is heavy-tailed or comprises outliers, we favor the Spearman's rank correlation over the traditional Pearson correlation. Moreover, this allows us to detect monotonic relationships beyond simple linear relationships. In our case, the values of each pair of features X and Y are converted to ranks r_X and r_y and the Spearman correlation coefficient ρ is computed as the covariance of the rank variables divided by the product of their standard deviations. Figure 5 shows the obtained coefficients for the nine numerical job and system features.

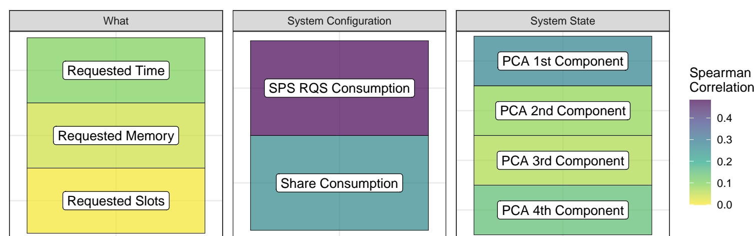

Fig. 5. Spearman correlation of numerical features with job wait time.

As expected, the job resource requests have almost no influence on job wait time in HTC datacenters. Conversely, the most impacting features are the system state and the current RQS and share consumption of the job owner's group. The other features have only a moderate impact.

Table 1. List of job and system features derived from the batch system logs and the analysis of intuitive causes.

	Name	Description	Transformation	Type
Who	Job Owner	Name of the user who submitted the job	Anonymization	Categorical
	User Group	Group to which the job owner belongs	Anonymization	Categorical
What	Requested Number of Slots	Number of cores/slots needed by the job	None	Numerical
	Requested Storage Subsystem	Job needs access to `SPS`, `HPSS`, or `iRODS`	None	Boolean
	Requested Time	Maximum CPU time (`[s,h]_cpu`) or runtime (`[s,h]_rt`). Default to the queue limits when no value is specified.	Fusion	Numerical
	Requested Memory	Maximum resident (`[s,h]_rss`) or virtual (`[s,h]_vmem`) memory. Default to the queue limits when no value is specified.	Fusion	Numerical
When	Submission Period	Period when the job is submitted (i.e., week {day, evening, night} and weekend {day, night})	Computed	Categorical
Where	Queue Name	Name of the submission queue	None	Categorical
System Configuration	Share Consumption	Current relative resource consumption of the user group at the submission time of the job	Computation	Numerical
	RQS Consumption	Current consumption of the Resource Quota Set of the user group at the submission time of the job	Computation	Numerical
System State	PCA Components	Number of jobs currently waiting to be executed or running at *group*, *queue*, and *global* levels	PCA	Numerical

4.2 Regression-based Correlation for all Features

We propose to determine the impact of the additional categorical and boolean features (i.e., job owner, user group, requested storage, submission period, and queue name), by computing their correlations with job wait time with a regression tree. Indeed, Pearson and Spearman correlation computation methods are respectively based on linear and isotonic regressions. The regression method consists in splitting our data set and trying to maximize the variance between the subsets. It is particularly suited for categorical variables and can also handle the other numerical features. Figure 6 presents the obtained correlation values with a regression tree of depth eight.

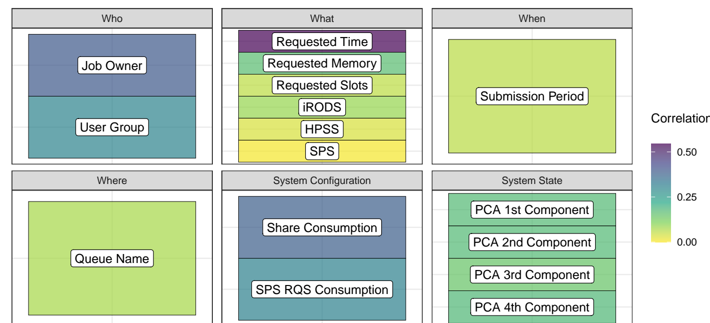

Fig. 6. Correlations of all features with job wait time.

Among the categorical features, the most influential category is related to who submitted the job. This is rather logical as the numerical feature with the highest Spearman's rank correlation is the RQS consumption that defines the most stringent constraints at group level.

A more surprising result is that *Requested Time* becomes the most influential feature according to this second correlation computation method, while its Spearman's rank correlation was pretty low. This can be explained by the limited number of requested time values that are provided by users. Indeed, each value corresponds to a small set of well-defined jobs always submitted by the same users with the same submission habits. Therefore, these are more likely to wait for similar amounts of time. A known drawback of this correlation computation method is that the algorithm learns on the same data it tries to predict. This causes an over-fitting of the regression which in turn leads to a good correlation factor for this feature. This is even amplified by users who do not indicate any particular requested time. In that case, the queue limits are used as default value and this information thus also includes knowledge about the submission queue and its relative priority. The observed high correlation between *Requested Time* and *Job Wait Time* may thus be artificial. However, cautiously using it in the training of our estimators can still be beneficial. While it could cause similar

biases for estimators based on a decision tree, this can also allow us to detect recurring behaviors related to certain users and leverage this knowledge to improve the prediction accuracy. This can be seen as a bad practice from a Machine Learning theory point of view but the practical interest cannot be neglected.

5 Learning-based Job Wait Time Estimators

This section presents how we apply Machine Learning techniques to pre-processed batch system logs to provide users with an estimation of the time their jobs will wait when they submit them. We first introduce the form taken by this estimation and the performance metrics used to estimate its quality in Section 5.1. Then we detail and discuss in Section 5.2 the ML algorithms considered to produce this estimation, while the obtained experimental results are presented in Section 5.3.

5.1 Objectives and Performance Metrics

Our first objective is to predict the time a job will wait as a single value based on the features of the newly submitted job and a training on a set of previously executed jobs for whom the wait time is known. However, this kind of estimation can quickly become inaccurate for several reasons. As shown in Fig. 1, about 56% of the jobs in the considered workload start less than 30 minutes after their submission and nearly 30% wait only for a few seconds. For such *quick starter* jobs [11] an estimation can easily be largely off without giving a useful information to the user. Moreover, the large number of features describing a job and the diversity of experienced wait times for jobs with similar features make it difficult to obtain a very accurate estimation. However, most users do not need an exact estimation of job wait time but would rather be interested in knowing a time range in which they can expect to see their job start. Then, we define in Table 2 eight wait time ranges. We also show the percentage of Local jobs belonging to each of these ranges in the original workload.

Table 2. Target wait time ranges.

Class	Wait Time Range	Workload fraction
1	Less than 30 minutes	56.21%
2	30 minutes to 2 hours	15.96%
3	2 hours to 4 hours	8.88%
4	4 hours to 6 hours	4.85%
5	6 hours to 9 hours	3.90%
6	9 hours to 12 hours	2.44%
7	12 hours to 24 hours	5.14%
8	more than 24 hours	2.61%

The first and dominant range corresponds to the fusion of the two leftmost regions of Fig. 1. Ranges 2 to 5 allow us to provide users whose jobs fall in the third region (30 minutes to 9 hours) with a more accurate estimation of the job wait time. Ranges 6 to 8 have the same purpose for jobs in the rightmost region that wait for more than nine hours.

To select the candidate ML algorithms to build our job wait time estimator, we rely on two complementary performance metrics. The *learning time* measures the time taken by an algorithm to process the entire set of historical data and build its prediction model. It mainly depends on the complexity of the algorithm itself and the size of the input data set. As the learning phase has to be done periodically to ensure keeping a good predictive power, this learning time has to remain reasonable. Then, we arbitrarily decided to set an upper limit to twenty four hours and discard candidate algorithms that need more time to learn. The *prediction time* is the time needed by an algorithm to infer the wait time of a newly submitted job. As we aim at returning the produced estimation to the user right after the submission of a job, we will favor the fastest algorithms and discard those taking more than two minutes to return an estimation.

To evaluate how close the predictions of the job wait time are to the observed values and be able to compare the accuracy of the considered ML algorithms we rely on the classical *Absolute Percentage Error* (APE) metric defined as $100 \times \frac{|y_i - \hat{y}_i|}{y_i}$ for $i \in 1, \ldots, n$ jobs, where y_i and $\hat{y}_i$ are the logged and predicted wait time of job i. However, this metric is known to be very sensitive to small logged data and to produce many outliers, so we will focus on the median and inter-quartile range of the distribution of this metric.

To evaluate the accuracy of the classification in wait time ranges, we analyze the confusion matrices produced by the different ML algorithms. A confusion matrix C is defined such that $C_{i,j}$ is the number of jobs whose wait time is known to be in range i and predicted to be in range j. We not only measure the percentage of jobs classified in the right range or in an adjacent range, but also the capacity of the algorithm to class jobs in every available ranges.

5.2 Job Wait Time Estimators

Our two objectives correspond to two classical applications of *Supervised Learning*. Estimating the exact time a newly submitted job will wait is a *regression* problem while determining to which wait time range it will belong is a typical *classification* problem. Multiple ML algorithms can be used to solve each of these two problems. They mainly differ from the tradeoff made between the time to produce a result and the accuracy of that result. In addition to the constraints on the learning and prediction times expressed in the previous section, the size of our data set raises another constraint on the amount of memory needed to perform the regression. This discards some methods such as the Logistic and Multivariate regression methods or the Least Absolute Shrinkage and Selection Operator (LASSO) [22] that would require several Terabytes of memory.

We selected four regression-based algorithms among the implementations made available by the Scikit-learn toolkit [16] to estimate the wait time of a

newly submitted job. The *linear regression* method is very fast to train and to return an estimation. However, it can be inaccurate when the relationship between features is not linear, which is the case for our data set. The *Decision Tree regression* [14] is a recursive partitioning method which is also fast but better handles data variability and categorical features. However, the depth of the tree has to be carefully chosen: if too shallow, the estimation will be inaccurate, while if too deep, there is a risk of data over-fitting. We also consider two ensemble learning methods which consist in combining several weak learners to achieve a better predictive performance. *AdaBoost* [7] starts by assigning an equal weight to the training data and computes a first probability distribution. It then iteratively boosts the weight of the mis-predicted instances to improve the accuracy of the final estimation. However, this method is known to be sensitive to noisy data and outliers. The *Bagging* [3] method consists in generating multiple training sets by uniform sampling with replacement of the initial data. Then, a weak learner is fitted for each of these sets and their results are aggregated to produce the final estimation. For both ensemble methods, we use a decision tree of variable depth as weak learner and split the training set into 50 subsets.

To solve our classification problem in wait time ranges, we follow two different approaches. The former consists in simply applying a classical classification algorithm to directly assign jobs in the different ranges from Table 2, while in the latter we first solve the regression problem of estimating the exact wait time of a job and then straightforwardly derive the wait time range for the job. We rely on the same families of algorithms as for solving the regression problem, i.e., Decision Tree, AdaBoost, and Bagging, but replace the basic Linear Regression method by a simple probabilistic classifier, i.e., Naive Bayes. For the Ensemble methods we use a Decision Tree classifier of variable depth as base estimator.

5.3 Experimental Evaluation

To evaluate the quality of a ML algorithm, the common approach is to split the initial data set into two parts. First, the algorithm has to be *trained* on a large fraction of the data set, typically 80%. Second, the *evaluation* of the performance of the algorithm is done on the remaining 20%. As our data are time-ordered, we cannot randomly pick jobs to build these two subsets. Indeed, when a new job enters the system, information is available only for jobs that have been *completed before* its submission. Consequently, our training set corresponds to the first 80% of the logs and the evaluation is done on the last 20%.

Leaning and Prediction Time We start by comparing in Fig. 7 the learning and prediction times of the candidate ML algorithms. For the Decision Tree and Ensemble methods, we consider different tree depths, while we have a single value for the Linear Regression and Naive Bayes approaches. We also distinguish the use of these algorithms to solve our regression and direct classification problems. These timings were obtained on one of the servers of the CC-IN2P3 equipped with a 40-cores Intel Xeon Silver 4114 CPU running at 2.20GHz.

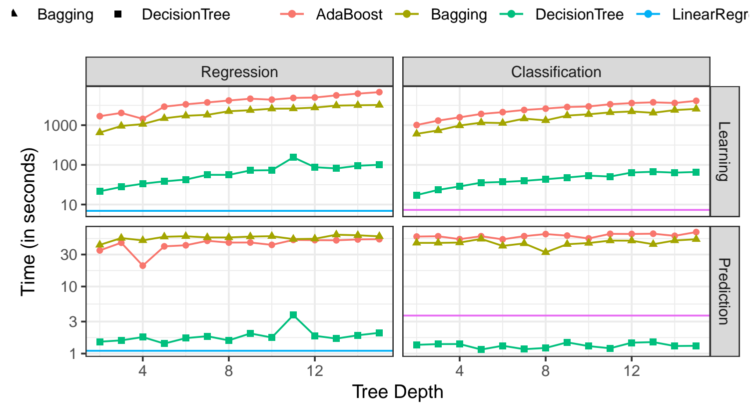

Fig. 7. Learning and prediction times of the different ML algorithms.

We can see that the Ensemble methods require much more time to be trained and produce an estimation than the basic regression and classification methods. The use of a deep Decision Tree is a good compromise with a training time of less than two minutes and only a few seconds to return a prediction.

Accuracy of Algorithms for the Regression Problem To evaluate the accuracy of the four ML algorithms used to solve the regression problem, we first have to determine what is the best tree depth for three of them. Figure 8 shows the evolution of the median Absolute Percentage Error with the depth of the tree. The dashed line corresponds to the accuracy of the Linear Regression.

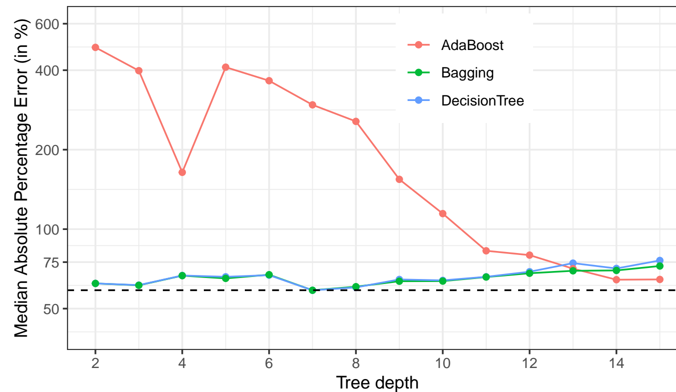

Fig. 8. Evolution of the Median Absolute Percentage Error with tree depth for jobs waiting for more than 30 minutes. The dashed line corresponds to the Median APE of the Linear Regression method.

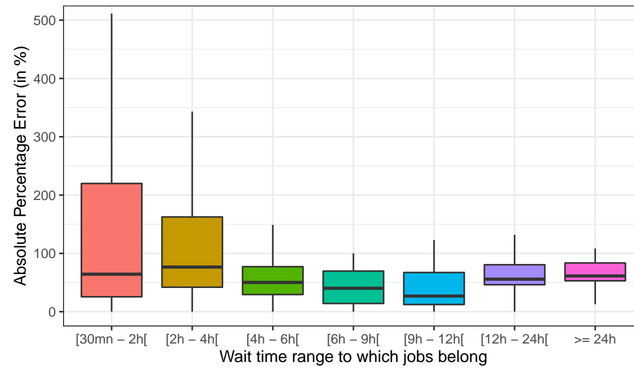

Fig. 9. Distribution of the Absolute Percentage Error achieved by a Decision Tree Regressor of depth 7 for jobs in different wait time ranges.

As APE leads to very large values for very short waiting times, the presented medians have been computed only for the jobs that waited for more than 30 minutes, i.e., about half of the initial workload.

The accuracy achieved by the AdaBoost algorithm is much worse than those of the others, because of its sensitivity to variable and noisy data. The Bagging and Decision Tree algorithms have very close accuracy and evolution along with the tree depth, but Decision Tree is much faster. We also see that the basic Linear Regression is on par with the more complex algorithms. However, we also measured the Absolute Error (in seconds) for the short-waiting jobs. We observed similar trends as in Fig. 8 for all algorithms but also a clear advantage of Decision Tree (median of 1h12 at depth 7) over Linear Regression (median of 2h20). Based on these results, we select a *Decision Tree Regressor of depth 7* as our regression-based job wait time estimator.

We complete this evaluation of the accuracy of the regression-based approach by analyzing the distribution of the APE in the different wait time ranges we target for the classification problem. Figure 9 shows this distribution for the selected Decision Tree Regressor of depth 7. We can see that the median APE is consistent across the different ranges and remains in the reasonable range of [26.8%; 76.6%]. We also see that the values of the third quartile and top whisker (i.e., 1.5 the inter-quartile range from the third quartile) tend to decrease as the job wait time grows. This means that the quality of the prediction becomes slightly better for long job wait times, i.e., where users need to know the most that they have to expect long delays.

Accuracy of Algorithms for the Classification Problem Table 3 evaluates the accuracy of the four direct classification methods we consider and that of the alternate approach that first solves a regression problem and then classifies the obtained predictions in wait time ranges. The accuracy is measured as the percentage of jobs that are classified either in the correct time range or in a directly adjacent one and the sum of these two values.

Table 3. Accuracy of the different classification algorithms.

Method	Correct	Adjacent	Combined
Naive Bayes	44.57%	19.55%	64.12%
Ada Boost (Depth 11)	44.81%	29.59%	74.40%
Decision Tree (Depth 9)	49.73%	19.55%	75.19%
Bagging (Depth 9)	47.08%	28.28%	75,36%
Reg. + Clas. (Depth 8)	22.26%	54.65%	76.92%

There is no clear winner among the four direct classifiers, with a difference of 5.16% of jobs classified in the right time range between the best (Decision Tree) and worst (Naive Bayes) algorithms. If we add the jobs classified in a directly adjacent time range, the Bagging algorithm becomes slightly better than the Decision Tree. The alternate approach (Reg. + Clas.) leads to more mixed results. It classes much less jobs in the correct time range but achieves the best percentage overall when including the adjacent ranges. However, these coarse grain results hide some important characteristics of the produced classifications. To better assess the respective quality of each algorithm, we study the confusion matrices for the different algorithms.

Figures 10 shows the confusion matrix for the Naive Bayes classifier and reads as follows. The tile on the sixth column of the third row indicates the percentage of jobs whose *measured* wait time was between 2 and 4 hours (third range) that have been classified in the [9h -12h[time range (sixth range). Then, the sum of the tiles on each row is 100%. The panel on the right of the confusion matrix indicates for each row the percentage of jobs that have been classified in the correct range or in an immediately adjacent one.

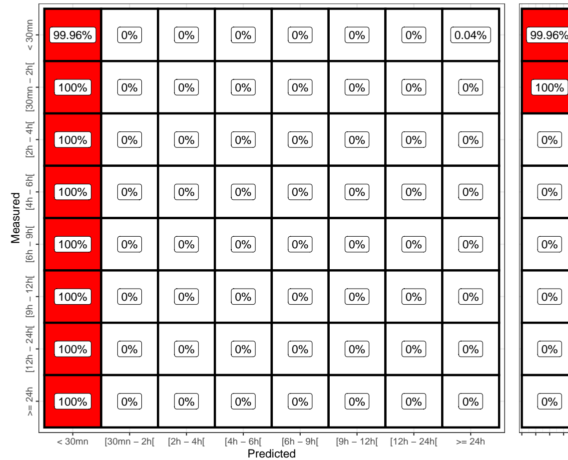

Fig. 10. Confusion matrix of the Naive Bayes classifier.

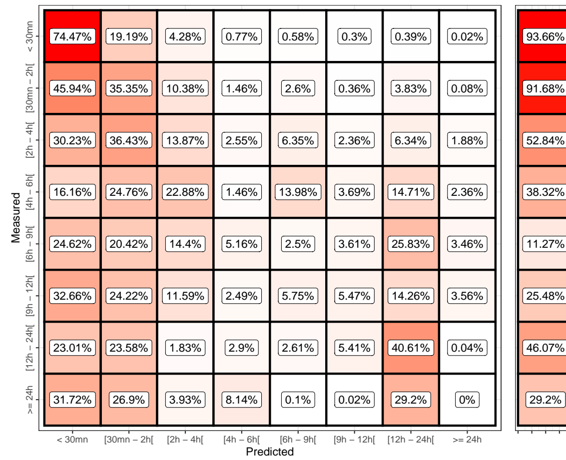

Fig. 11. Confusion matrix of the AdaBoost classifier.

We can see that almost all the jobs are classified in the first time range (i.e., less than 30mn). Then, the apparent good results of this method in Table 3 actually mean that Naive Bayes correctly classifies only jobs in that first range, which are dominant in our test set, but fails for all the jobs in other time ranges.

Figure 11 shows that while not being as binary as Naive Bayes, the AdaBoost classifier suffers from a similar bias and tends to classify too many jobs in one of the first two ranges (i.e., less than 30mn and from 30mn to 2 hours). In other words, this methods largely underestimates the wait time for most of the jobs. Then, the percentage of jobs classified in the right time range or an adjacent one clearly drops as soon as jobs wait for more than two hours.

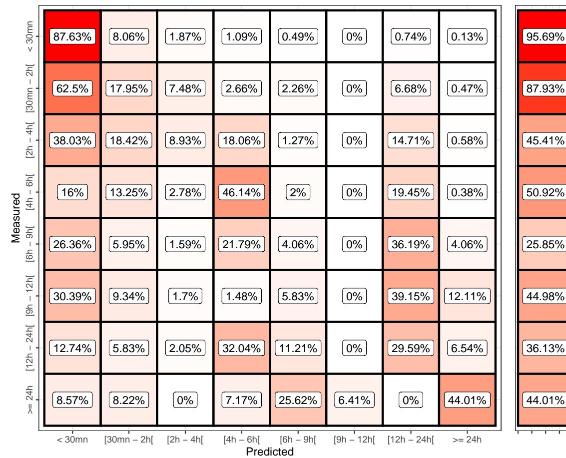

Fig. 12. Confusion matrix of the Decision Tree classifier at depth 9.

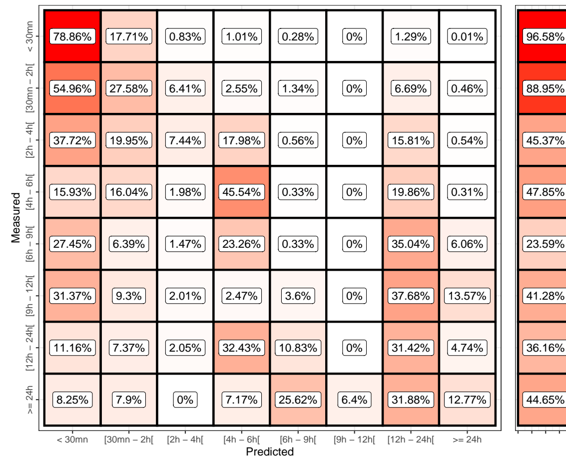

Fig. 13. Confusion matrix of the Bagging classifier at depth 9.

The first two columns of the confusion matrices for the Decision Tree (Fig. 12) and Bagging (Fig. 13) classifiers also show that they greatly underestimate the wait time of many jobs. However, they lead to slightly better predictions than AdaBoost as they are both able to correctly classify more jobs in the fourth (4 to 6 hours) and eighth (more than 24 hours) ranges.

The last confusion matrix shown in Fig. 14 is that obtained for when we classify the predictions made by a Decision Tree Regressor of depth 8. First, it explains why this approach leads to only 22.26% of correct predictions overall, while the direct classifiers are above 44%. This is mainly caused by the results for the shortest time range. Most of the jobs that actually waited for less than 30

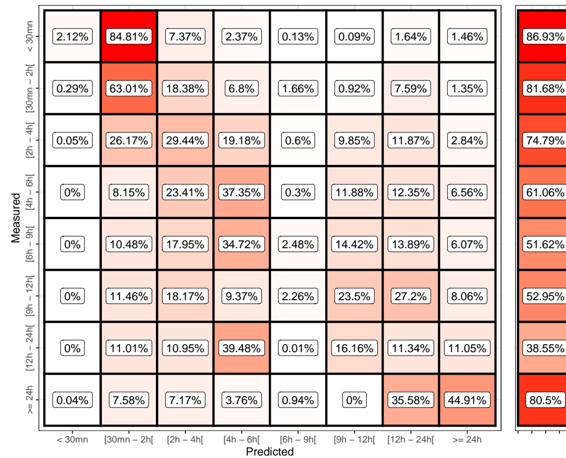

Fig. 14. Confusion matrix of the Regression + Classification approach.

minutes are predicted to wait between 30 minutes and two hours, hence classified in the second range. We further analyzed the distribution of the predicted job wait time in that particular time range and saw that nearly half of these jobs were predicted to wait for only 38 minutes.

Another main difference of this confusion matrix with those of the direct classifiers is that a very small number of jobs are classified in the first range. Instead, we observe a better distribution of the predictions along the diagonal of the matrix. A direct consequence is that the percentages of jobs in the correct time range or an adjacent one, shown by the right panel, are much better and more stable with this method.

To summarize, when the proposed *Regression + Classification* approach fails to predict the right time range, it generally deviates from only one class, hence the best overall accuracy in Tab. 3. We thus propose to use this original method to implement a job wait time prediction service for the users of the CC-IN2P3.

6 Applicability to Other Workloads

The work presented in this paper is tightly coupled to a specific workload processed at a specific computing center. One can thus ask whether the proposed methodology and presented results can be transposed to another computing center. We discuss this legitimate question in this section.

The IN2P3 Computing Center is a large scale *High Throughput Computing Center* implementing a *Fair-share* scheduling algorithm configured with *scheduling queues*, *resource quotas,* and *priorities* that processes a workload composed of a vast majority of *single-core* jobs with a *utilization* close to 100%. Note that such a settings is at least representative of the twelve other large computing centers worldwide involved in the processing of the data produced by the LHC and the rest of the associated computing grid. This represents hundreds of smaller scale computing centers dealing with similar workloads and sharing similar configurations of their infrastructures and working habits. The proposed study can thus be rather straightforwardly applied to these computing centers.

Going from a High Throughput Computing Center to a High Performance Computing Center will have an impact on the typology of the submitted jobs, i.e., more parallel jobs, but also on the scheduling algorithm, i.e., EASY/Conservative backfilling instead of Fair-Share. We already mentioned in Sect. 4 that it will change the correlation between associated resource requests and job wait time, hence impact the results. However, some features will remain very similar, such as the difference of usage and priorities between user groups, submission patterns [6], or the influence of quotas or of the current state of the system.

We believe that these similarities and differences do not compromise the soundness of the methodology we followed in this work, which can be applied to any workload or configuration: i) **Analyze** the workload and **review** the system configuration; ii) **Identify** what could be intuitive causes for a job to wait; iii) **Confirm** these intuitions by determining correlations of job and system features with job wait time; iv) **Compare** the time to learn and predict and the accuracy

of several ML algorithms; and v) **Question** the obtained results by looking at them from different angles. To ensure the reproduction and further investigation of the presented results, as well as the adaptation of the proposed methodology to another context, we prepared an experimental artifact that comprises the anonymized batch logs and metadata, and all the data preparation scripts, calls to ML algorithms, and obtained results in a companion document [8].

7 Conclusions and Future Work

The Quality of Service experienced by users of a batch system mainly depends on the time submitted jobs wait before being executed. However, in High Throughput Computing datacenters whose batch system implements the Fair-Share scheduling algorithm, this job wait time can quickly become unpredictable and range from a few seconds to several hours, or even days in some pathological cases. Therefore, we proposed in this article to leverage Machine Learning techniques to provide users with an estimation of job wait time. To this end, we analyzed 23 weeks of logs of the batch system of the IN2P3 Computing Center, the main French academic HTC datacenter. First, we identified some intuitive causes of the job wait time and determined its formal correlation with sixteen job and system features. Then, we compared four Machine Learning algorithms to either determine an estimation of the wait time or a wait time range for each newly submitted job.

We found out that the best tradeoff between learning and prediction times and accuracy to solve the regression problem of estimating the job wait time was achieved by a *Decision Tree Regressor of depth seven*. It allows us to obtain consistent and relatively good estimations in only a few seconds, hence enabling a direct return to users at submission time. We also observed that the direct classification of jobs in wait time ranges led to imperfect results, with a severe underestimation of the wait time for a majority of jobs. The imbalance of the workload with more than 50% of *quick starter* jobs may be the main cause of the observed results by creating a bias in the training of the algorithm. However, we also showed that classifying the jobs thanks to a regression-based estimation of their wait time led to more promising results with nearly 77% of the jobs assigned in the right wait time range or in an immediately adjacent one.

As future work, our first task will be to apply the proposed methodology to other workloads, either HTC or HPC, to assess its robustness. Then we plan to refine our classification algorithms to account for the heterogeneity of the distribution of training jobs in the different time ranges. Thanks to an initial fast clustering of the training set, we should be able to derive a weighting scheme for the target classes and thus improve the quality of predictions. We also plan to design an original learning-based estimator that would leverage the expertise of both the batch system operators and members of user groups. The idea is to build on the set of intuitive causes identified in Section 3 to derive a new set of features on which to train the algorithms. For instance we could include more information about the facility internal policies (e.g., job boost after maintenance,

or penalties for bad usage) or annual/seasonal patterns (e.g., paper or experiment deadlines). Finally, we will investigate the use of Deep Learning methods to solve this job wait time prediction problem.

On a more practical side, we plan to transfer the tools developed during this study to the team in charge of user support at CC-IN2P3. The objective is to integrate a new service letting users know how much time their jobs are likely to wait once submitted to the recently deployed user portal of the computing center. As part of this effort, we also aim at determining the minimal size of the training set that would allow us to obtain accurate enough results. Indeed, the training of the ML algorithms has to be done again periodically to account for variations in the workload or changes of the infrastructure. Reducing the size of the training set will reduce the onus on the support team and allow for better reactivity. Finally, we aim at providing users with some feedback on their submission habits in order to help them reduce the wait time of their jobs.

Acknowledgments

The authors would like to thank Wataru Takase and his colleagues from the Japanese High Energy Accelerator Research Organization (KEK) for providing the initial motivation for this work.

References

1. Azevedo, F., Gombert, L., Suter, F.: Reducing the Human-in-the-Loop Component of the Scheduling of Large HTC Workloads. In: Proc. of the 22nd Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 11332, pp. 39–60. Vancouver, Canada (May 2018). https://doi.org/10.1007/978-3-030-10632-4_3
2. Azevedo, F., Klusáček, D., Suter, F.: Improving Fairness in a Large Scale HTC System Through Workload Analysis and Simulation. In: Proc. of the 25th International Euro-Par Conference (Euro-Par). LNCS, vol. 11725, pp. 129–141. Göttigen, Germany (Aug 2019). https://doi.org/10.1007/978-3-030-29400-7_10
3. Breiman, L.: Stacked Regressions. Machine Learning **24**(1), 49–64 (1996). https://doi.org/10.1007/BF00117832
4. Brevik, J., Nurmi, D., Wolski, R.: Predicting Bounds on Queuing Delay for Batch-Scheduled Parallel Machines. In: Proc. of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP). pp. 110–118. New York, NY (Mar 2006). https://doi.org/10.1145/1122971.1122989
5. Feitelson, D.: Metrics for Parallel Job Scheduling and Their Convergence. In: Proc. of the 18th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 2221, pp. 188–206. Cambridge, MA (Jun 2001). https://doi.org/10.1007/3-540-45540-X_11
6. Feitelson, D., Tsafrir, D., Krakov, D.: Experience with using the Parallel Workloads Archive. Journal of Parallel and Distributed Computing **74**(10), 2967–2982 (2014)
7. Freund, Y., Schapire, R.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and Systems Sciences **55**(1), 119–139 (1997). https://doi.org/10.1006/jcss.1997.1504

8. Gombert, L., Suter, F.: Companion of the "Learning-based Approaches to Estimate Job Wait Time in HTC Datacenters" article (2021), Available at: https://doi.org/10.6084/m9.figshare.13913912
9. Jancauskas, V., Piontek, T., Kopta, P., Bosak, B.: Predicting Queue Wait Time Probabilities for Multi-Scale Computing. Philosophical Transactions of the Royal Society A **377**(2142) (2019). https://doi.org/10.1098/rsta.2018.0151
10. Kay, J., Lauder, P.: A Fair Share Scheduler. Communications of the ACM **31**(1), 44–55 (Jan 1988)
11. Kumar, R., Vadhiyar, S.: Prediction of Queue Waiting Times for Metascheduling on Parallel Batch Systems. In: Proc. of the 18th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 8828, pp. 108–128. Phoenix, AZ (May 2014). https://doi.org/10.1007/978-3-319-15789-4_7
12. Li, H., Chen, J., Tao, Y., Groep, D., Wolters, L.: Improving a Local Learning Technique for QueueWait Time Predictions. In: Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid). pp. 335–342. Singapore (May 2006). https://doi.org/10.1109/CCGRID.2006.57
13. Li, H., Groep, D., Wolters, L.: Efficient Response Time Predictions by Exploiting Application and Resource State Similarities. In: Proc. of the 6th IEEE/ACM International Conference on Grid Computing (GRID). pp. 234–241. Seattle, WA (Nov 2005). https://doi.org/10.1109/GRID.2005.1542747
14. Loh, W.Y.: Classification and Regression Trees. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **1**, 14–23 (2011). https://doi.org/10.1002/widm.8
15. Mu'alem, A., Feitelson, D.: Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. IEEE TPDS **12**(6), 529–543 (Jun 2001)
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
17. Schlagkamp, S., Ferreira da Silva, R., Allcock, W., Deelman, E., Schwiegelshohn, U.: Consecutive Job Submission Behavior at Mira Supercomputer. In: Proc. of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC). pp. 93–96. Kyoto, Japan (May 2016). https://doi.org/10.1145/2907294.2907314
18. Smith, W.: A Service for Queue Prediction and Job Statistics. In: Proc. of the 2010 Gateway Computing Environments Workshop. pp. 1–8. Los Alamitos, CA (Nov 2010). https://doi.org/10.1109/GCE.2010.5676119
19. Smith, W., Foster, I., Taylor, V.: Predicting Application Run Times with Historical Information. JPDC **64**(9), 1007–1016 (2004). https://doi.org/10.1016/j.jpdc.2004.06.008
20. Smith, W., Taylor, V., Foster, I.: Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In: Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing. LNCS, vol. 1659, pp. 202–219. San Juan, Puerto Rico (Apr 1999). https://doi.org/10.1007/3-540-47954-6_11
21. The IN2P3 / CNRS Computing Center: http://cc.in2p3.fr/en/
22. Tibshirani, R.: Regression Shrinkage and Selection via the Lasso. Journal of the Royal Statistical Society. Series B (Methodological) **58**(1), 267–288 (1996). https://doi.org/10.2307/2346178
23. Univa Corporation: Grid Engine. http://www.univa.com/products/