Machine Learning-Based Predictions for Optimal Job Scheduling

Amy He, Heather Han, Jennifer Baron, Sharmila Tamby

Department of Computer Science
Johns Hopkins University
Baltimore, MD

ABSTRACT

Many big data systems today must process large amounts of data in an inexpensive and efficient way. Effectively scheduling jobs to achieve high performance is currently an open area of research. Specifically, predicting the resources needed for jobs is a key challenge for schedulers to maximize resource utilization and minimize job completion times. Current predictors lack precise information regarding the requirements of the services [1]. We built a machine learning model to capture job information and dynamically predict the CPU resource a job may need. Traditional methods estimating resources statically prior to the job results in inefficient resource allocation because the amount of available resources changes as jobs enter and exit the cluster [2]. Therefore, the model also incorporates the concept of progressive deadlines by modularizing each job and accounting for the amount of resources that other jobs are occupying. Our evaluation shows that to improve scheduling performance, machine learning 1) can provide accurate resource predictions with < 10% error and 2) has the potential to capture specific information about cluster environments for progressive deadlines.

1. INTRODUCTION

Many big data systems employ simple job schedulers with varying heuristics. Big data systems have to schedule many tasks but have limited resources available to schedule these tasks. These limited resources include processors, memory, disk, and network. A growing area of research is how to ensure a high utilization of resources and low latency.

Traditionally, scheduling algorithms such as Earliest Deadline First (EDF), Shortest Job First (SJF) and First in, First out (FIFO) are deployed in big data systems. In all of these cases, the predicting resources needed to perform the task is a key challenge for the algorithms to perform optimally. If the provider does not have precise information regarding the requirements of the services, this leads to an inefficient resource allocation. For example, EDF gives priority to tasks with the earliest deadline and will execute those tasks first [1]. EDF works well under low or moderate levels of resource contention, but then worsens in an overloaded system. This is because tasks only gain high priority when they are close to their deadlines, which can cause a domino effect of other jobs missing their deadlines. Additionally, EDF without proper resource prediction may discard jobs when its deadline has passed, which leads to a potential waste of resources. Similarly, SJF is a scheduling policy which selects the waiting policy with the smallest execution time to execute next [8]. SJF performs well when it makes the strong assumption that true run times are known in advance, which is not usually the case because the estimations do not capture proper information about the submitted job. Apache Hadoop, widely used to process big data, uses FIFO scheduling policy by default, in which jobs are processed in the order that they arrive [5]. Following this policy, if longer jobs are submitted first and gain higher priority than shorter ones submitted later, then starvation may occur when the shorter jobs never reach completion.

The problem with these traditional scheduling algorithms is that they do not predict the resources each job needs. This limits the capability of running computationally intensive tasks due to potential starvation and high latency. If a scheduling algorithm is capable of ordering jobs optimally, we hypothesize this would minimize missing deadlines and job starvation [7].

Therefore, this project proposes an innovative solution to dynamically predict resources of a job to assist in optimal job scheduling. First, literature shows that many different scheduling heuristics use time as a measure to predict various tasks [4]. Thus, predicting the CPU time of jobs was chosen as the main focal point of this project. Using CPU time as a resource will enable more intelligent scheduling for schemes that are reliant on the progress of jobs since it will look into resources that each task will need.

Next, machine learning was leveraged to aid in efficient job scheduling. Machine learning models have been successfully applied to prediction problems because they are capable of learning anomalous patterns, leveraging past knowledge of users and their behavior, and are robust to noise and human error that could occur when submitting jobs. In this

case, machine learning models learn resource requirements and base predictions off of a feature space with information on task commands and input files.

Finally, progressive deadlines are used. The concept of progressive deadlines has been proven to minimize overall loss in server-side scheduling for jobs with consecutive deadlines [2]. Instead of scheduling an entire job once before entering the workflow, scheduling and resource allocation can instead be performed on a more specific task level, in which one job can have many tasks. This way, even if one piece of a job is not completed by its deadline and is lost, the subsequent pieces of the job still have the potential to be delivered on time. Making predictions on a modular level and using multiple deadlines corresponding to the progress of jobs mitigates issues that arise with the dynamic resource demands of tasks and jobs in a workflow.

This project integrates the novel approaches of both machine learning and progressive deadlines to dynamically predict CPU resources and improve resource prediction models.

2. RELATED WORK

Currently, a resource such as time is traditionally predicted using methods such as the exponential averaging method [3]. This is a mathematical formula that depends on using the process' previous history to predict job length [10]. However, literature shows that the results are not always reliable or accurate using this method, because a mathematical formula has a tendency to smooth the trendline, thus lowering the precision of predictions [9]. Another problem is that these traditionally make static predictions, in which resources are predicted once before entering the workflow, to determine priority during the scheduling process. This is problematic because optimal resource allocation may change dynamically and static predictions do not account for how jobs that are currently running may interact with each other.

A number of machine learning frameworks have been designed to improve job scheduling algorithms [9]. Previous studies show that characteristics of a process contributed significantly to the prediction of required resources. Previous executions of a job were analyzed to evaluate values for chosen attributes of a process. Decision Trees, K-NN, and Decision Tables were used to predict "total execution time", then important attributes were evaluated and ranked according to importance using various search methods, such as Genetic Search, Best First Search and Rank Search. Overall, while various attempts have been made to improve job schedulers with machine learning, these methods have the same weakness of static predictions.

Moreover, progressive deadlines have been explored in the context of server-side scheduling for video service. In an environment with requests from multiple clients, in which each request has a deadline, the goal of real-time scheduling is to minimize job completion time and to complete each request by its deadline. If the server has limited capacity, any content not delivered by the deadline is lost. Thus, employing progressive deadlines in this environment allowed the server to treat packets from a single job as independent jobs. If a portion of a job is not completed by its deadline and ultimately drops, subsequent portions will continue to be delivered.

Overall, while there have been many investigations to improve job scheduling algorithms, the resource prediction models typically do not take advantage of machine learning, and many cannot accommodate the dynamic allocation of resources.

3. METHODS

3.1 Selecting Datasets

Various datasets were compiled from Kaggle, a data science tool that aggregates and publish data, to use as the sample space. 52 datasets, ranging from 25KB to 1GB, were downloaded using the Kaggle API. Datasets were then concatenated to generate over 100 larger datasets from 0.1 MB to 20GB by randomly duplicating the 52 Kaggle datasets.

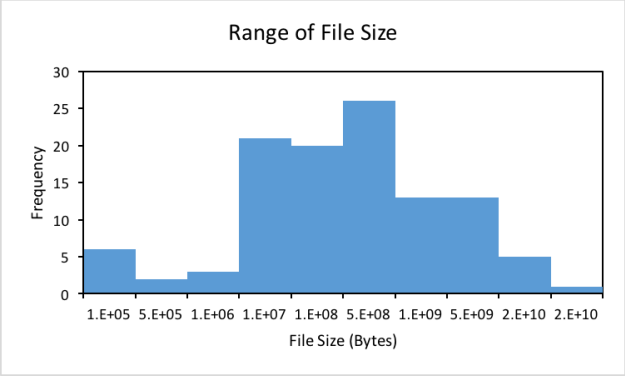

Figure 1: Histogram visualizing the range of sizes for each dataset used as input for jobs. File size in bytes is a key feature for the machine learning predictions.

3.2 Generating Scripts

After generating datasets, we generated 100 scripts to use as our sample space. Each script has 4-7 random combinations of commands, including sort, maximum, minimum on random columns and grep on random phrases. 70 of these scripts randomly selected from 70% of the generated files are used as input to train the machine learning model. The rest of the scripts used the remaining generated files as input and are used in the testing phase. The features for each script are the number of a certain command, size of the input file on which a certain command will be running, and the number of lines a certain command will be running.

3.3 Selecting Features

The feature space was generated according to file characteristics that are correlated with job completion times. Figures 2, 3, and 4 show how file size is correlated time to sort, time to identify a maximum variable, and time to find the minimum element in the dataset. There is a 0.93 correlation between file size and sort time, 0.98 correlation between file size and max time, and a 0.98 correlation between file size and min time. This ascertains that file size is a useful characteristic in predicting time.

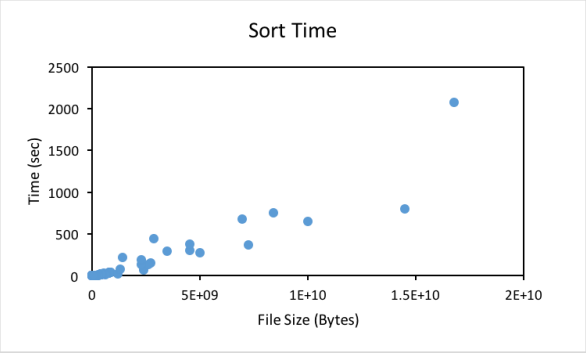

Figure 2: Time to sort dataset plotted against varying file input sizes.

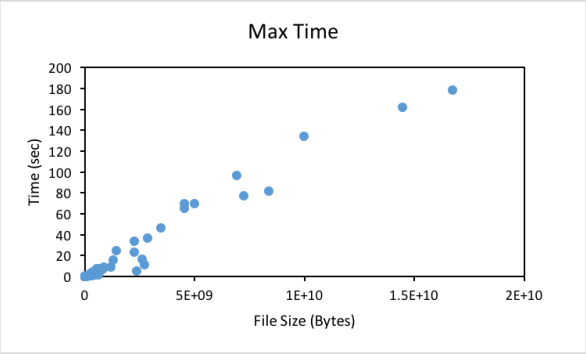

Figure 3: Time to find maximum element in dataset plotted against varying file input sizes.

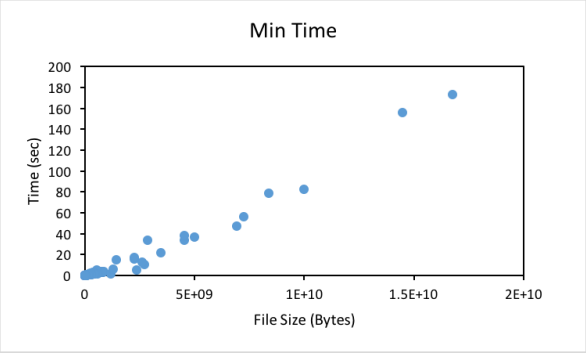

Figure 4: Time to find minimum element in dataset plotted against varying file input sizes.

3.4 Building Machine Learning Models

After building the sample and feature spaces, we built a linear regression and a random forest regression model. We chose to start with linear regression due to its interpretability, in which we can clearly evaluate how well the feature space captures information on CPU resource requirements. We also chose to implement random forest because of its ability to work well on a variety of datasets. Both models were trained with train scripts and their respective features and resource labels. Then, each model predicted CPU resource for the test scripts given their respective features.

3.5 Running Scripts

To simulate a job scheduling environment, Google Cloud Dataproc was used for running Apache Hadoop clusters. Each script was submitted to the cluster and has a real label of the CPU resource needed. The model's predicted labels and Dataproc's real labels of CPU resource are then compared in order to analyze the performance of the machine learning model.

Progressive deadlines require multiple predictions throughout a job that take into account the changing environment in which the job is running. Therefore, the scripts were submitted to Dataproc in various environments, such as when only one other job is asynchronously using resources or when six other jobs are asynchronously using resources. The feature set takes into account the cluster workload to predict CPU resources by keeping a feature corresponding to the number of asynchronous jobs running at a given time. For the purposes of the project, a jobs is simulated as multiple sequential scripts, with each script ran as an individual Hadoop job. By making a prediction on each script, specifically each "task" in a job, we can account for a dynamically changing environment. This will allow for evaluation of whether making modular and environment - sensitive predictions in a machine learning model can help accomplish progressive deadlines. These predictions that account for modularity and workload are then compared to those that do not.

3.6 Evaluating Models

To evaluate model performance against non-progressive jobs, a simulated "non-progressive" test set of jobs was generated. The non-progressive train and test sets used the same scripts and features as the timed progressive jobs, with the only feature for each script changed being the number of asynchronous jobs running simultaneously. For a non- progressive job, we assumed that only the initial state will be known, and the runtime is only predicted once at the beginning of the job. To model this, we took the number of asynchronous scripts running during the first script in the job and assigned that number to the subsequent scripts in the job as a feature. Each script was randomly assigned as a task to different jobs. The same method of assigning scripts to jobs was also performed for the train and test sets of progressive deadline jobs, which takes into account the varying number of asynchronous scripts over time, for the sake of evaluation

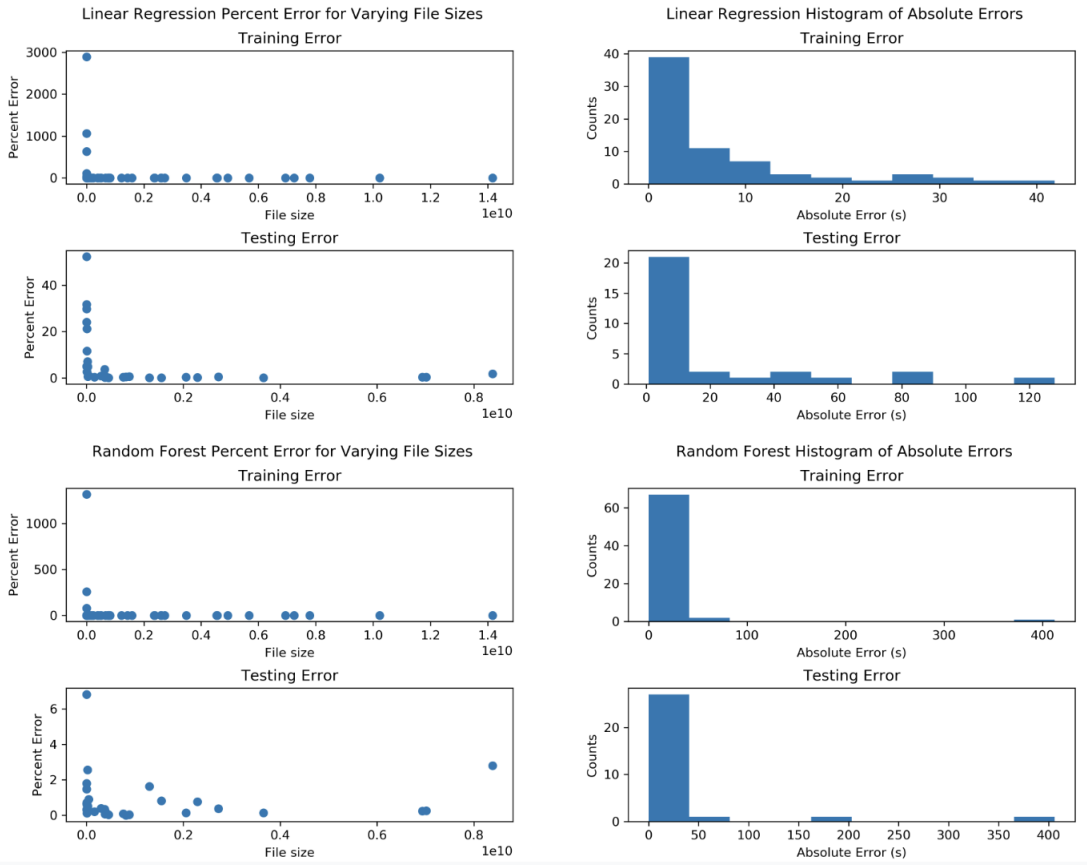

Figure 5: Error plots reflecting Linear and Tree Regression model performance on 100 scripts each with 1-3 commands. File size is measured in bytes and error is measured in seconds.

against the non-progressive performance. Models were then trained on the progressive train set of jobs, and tested on both the progressive and non-progressive test sets.

Overall, we built regression models to demonstrate as a proof of concept that machine learning is an effective technique for predicting length of tasks, including when each job is modularized into separate tasks (or scripts) to implement progressive deadlines.

4. RESULTS

Previous studies of this kind utilize error plots to evaluate model performance [7]. Specifically, the usefulness of the predictions are evaluated using mean absolute error and percent errors to show the error distribution. Similarly, this project evaluates the performance of the machine learning models using training and test sets of jobs. The percent error as well as the absolute errors for each script is measured. This metric evaluates how close the predicted CPU resource is to the true CPU resource.

4.1 Modeling Scripts With 1-3 Commands

We first tested model performance using single commands with random datasets, and the regression models performed moderately well on each commands. The percent error varied with file size, with the largest percent error corresponding with the smaller input files. Then we added multi-commands for each script in order to simulate more realistic scripts. The addition of command-specific features were able to capture enough information to make good predictions (Figure 5). Though we continue to see a higher percent error for small files, the absolute error corresponding to it is low as most of them are below 100 seconds. We also see a higher performance in Random Forest compared to Linear Regression, as expected from a model known to have more predictive power than a simple linear regression model.

4.2 Modeling Scripts With 4-7 Commands and Varying Asynchronous Workloads

Then, model performance was tested on more complex scripts, with a higher number of 4-7 commands each and submitted during different cluster workload environments. Therefore, a script may be submitted while three other background scripts are running and the same script may be submitted while seven other background scripts are running. Each of these new scripts may take several minutes to hours to run. After creating a model with features sensitive to cluster workload, the performance was evaluated again for both linear and tree regression models, as seen in Figure 6. The

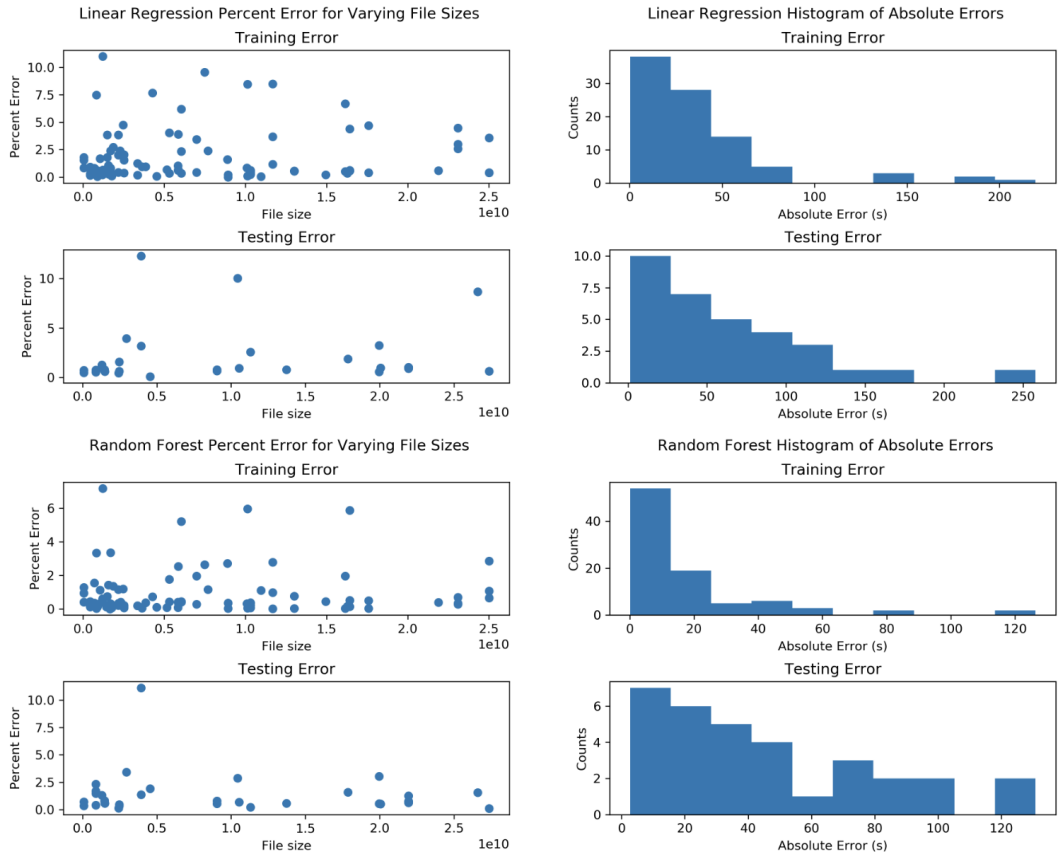

Figure 6: Error plots reflecting Linear and Tree Regression model performance on scripts each with 4-7 commands, where scripts were submitted multiple times, each time in a different cluster workload environment. File size is measured in bytes and error is measured in seconds.

percent error is now more scattered and less sensitive to total file input size and has consistently remained under 10%. Though the runtimes of each script are significantly longer, most absolute error remains under 100 seconds. There is still a slight improvement seen in tree regression compared to linear regression.

4.3 Evaluating progressive vs. non-progressive prediction performance

If progressive features are not taken into account, a job of scripts submitted would make a single static prediction before it enters the cluster. For instance, if Script A, B, and C are in a job, the predictions for B and C are made in the context of A's cluster environment because A is the first script to run in the job. If A was submitted while three other asynchronous jobs are running in the cluster, then B and C have predicted runtimes in the context of three background jobs as well. This serves to simulate a static, non-progressive, prediction once at the beginning of the job. In Figure 7, performance of this static prediction is compared to the true CPU time, where each script runs with different number of asynchronous jobs in the background. Performance of dynamic prediction where each script has progressive features taken into account is also compared to the true CPU time in Figure 7. There was no significant difference between the model performance with and without the feature of number of background jobs simultaneously running with each script because the absolute error distributions were quite similar.

5. DISCUSSION

Overall, the machine learning models do not reveal a clear distinction between dynamically predicting progressive deadlines and using static non-progressive deadlines. This is likely because the feature set was not comprehensive enough to capture all differences between the progressive and non- progressive train and test sets of jobs, leading to a less robust model. The main feature used for predicting CPU time of progressive deadlines was the number of asynchronous jobs simultaneously running in the background. We postulated that this should be a key feature for predicting the CPU time of progressive jobs, as it conveys a notion of the amount of compute resources such as processors, memory, disk, and network available for each task in the job. However, the results of the models show that this single feature is not sufficient for modeling progressive jobs.

When evaluating the model, we considered several evaluation metrics, particularly those suggested by the TAs, such

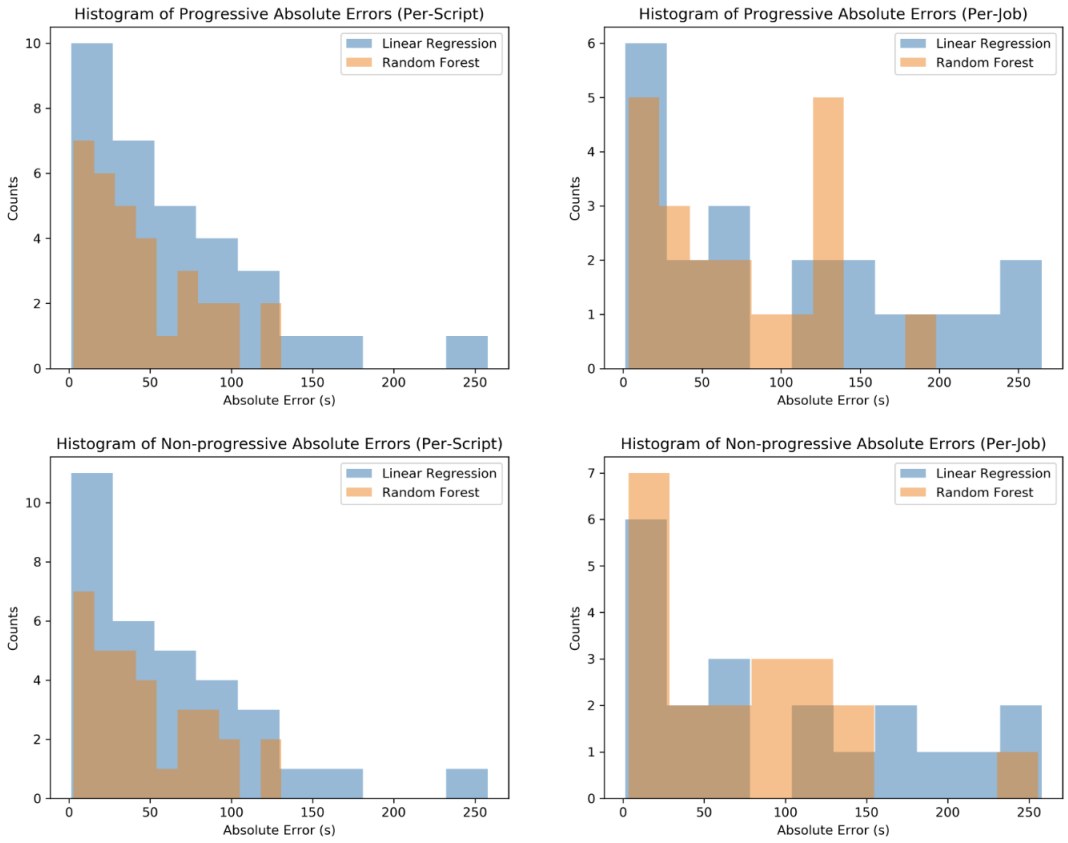

Figure 7: Histogram of absolute errors on scripts, where each script belongs to a certain job. The cluster workload feature for all the scripts is the first script's environment in a non-progressive job, and script-specific in a progressive job.

as ROC curves. However, because the label space is not categorical, it was difficult to define correct interpretations of "false positive" and "false negative" in our evaluation [6]. Thus, we chose to evaluate our results using absolute and percent error, with a baseline metric of timed progressive tasks.

Despite the similarity in performance of the machine learning models on predicting CPU time of non-progressive and progressive tasks and jobs, the overall percent error and absolute error of the model were low. This implies that the other features used for the model, such as the number of sort commands and the input file size, were predictive of CPU time. The strong dependence of the model on file size and these other features also suggest that one reason for a lack of significant difference in predictions on progressive and non-progressive test sets could be the lack of special attention given to the number of asynchronous tasks running in the background. As no feature weighting is being performed, all features are treated as equally important and no extra importance is placed on features more relevant for progressive deadlines. The low percent errors confirm that machine learning is a viable option for predicting CPU time; however, as a next step, this primitive model can be further developed by incorporating and weighing additional features to build a more robust machine learning model.

6. LIMITATIONS

One limitation is that the model simulated jobs that contained basic commands- sort, maximum, minimum, and grep. The jobs are not reflective of more complex jobs that larger corporations in a real-world environment run on their big data processing systems. Nonetheless, our machine learning models were simple regression models and despite its lightweight and simple context, they were able to perform well on varying combinations of commands, input files, and cluster workload. By using more developed machine learning algorithms, they can capture the level of complexity of organizations‘ workflows.

Additionally, this work does not involve creating a new job scheduler with machine learning and progressive deadlines incorporated into its algorithm. Rather, this is an experiment to prove that already existing job scheduling algorithms can be made more efficient and robust using the aforementioned machine learning methods. In order to understand the performance of good resource predictors incorporated into the decision-making process of job scheduler, further experiments are needed. There may be concerns that incorporating a predictive model may be computationally intensive and

cause latency in schedulers if used in practice.

Finally, our work to compare progressive and non-progressive deadline pushes for machine learning algorithms, because they are able to learn the the environment specifics of each job across time. However, our work simulated a changing environment by submitting different number of tasks asynchronously, which may not reflect all the changes and complexities that predictive models should capture of a true environment as tasks are completed and jobs exit and enter a workflow.

7. FUTURE WORK

To improve the robustness of the models, additional approaches can be experimented with. First, cross validation would be a useful tool to validate our results. Furthermore, the true run times were run on the default cluster configuration on Google Cloud Dataproc. A future investigation would be to experiment whether predictive models can be sensitive to different cluster configurations, such as the number of workers in the cluster, the size of the cluster, and different operating systems. Additionally, real-world environments may submit jobs with datasets significantly larger than 20 GB. Datasets such as genomic data can be incorporated to understand the performance on substantial datasets. Besides adding additional features and experimenting with more complex data, machine learning models with more power can be implemented. Specifically, both linear regression and tree regression models require a preprocessed feature space. Utilizing neural networks would be an interesting future investigation due to its ability to process raw data (jobs and datasets) and to weigh important features and approximate any nonlinear function on its own.

Finally, a prototype of a scheduling algorithm such as SJF can be adjusted to include our methods of machine learning and progressive deadlines. SJF depends on recognizing the shortest job, so resource utilization and predicted time for job completion can then be estimated and evaluated against traditional SJF.

8. CONCLUSION

Overall, our work utilizes both machine learning and progressive deadlines to dynamically predict CPU resources. Random Forest consistently outperforms the Linear Regression model. However, both models successfully minimize absolute error. We also found that utilizing number of asynchronous jobs as a feature did not adequately represent the cluster environment, thus resulting in similar error rates across various machine learning models in both progressive and non-progressive configurations. Although additional metrics are needed to fully capture dynamic environments, the simple models that we built were still able to perform well and adequately predict CPU time. We show that understanding the dynamic resource needs of submitted jobs is a robust method to assist scheduling algorithms in making informative and accurate decisions.

Acknowledgments

Thank you to Dr. Soudeh Ghorbani and the teaching assistants of Cloud Computing for project guidance.

Link to code repository:

`https://github.com/jfb843/CloudComputing`

9. REFERENCES

[1] V. B. Anderson, James H. and U. C. Devi. An edf-based scheduling algorithm for multiprocessor soft real-time systems. In $17thEuromicroConferenceonReal-TimeSystems(ECRTS'05)$, 2005.

[2] S. B. Gardner, Kristen and M. Harchol-Balter. Optimal scheduling for jobs with progressive deadlines. In *IEEE Conference on Computer Communications (INFOCOM)*, 2015.

[3] S. A.-A. Helmy, Tarek and O. Bin-Obaidellah. A machine learning-based approach to estimate the cpu-burst time for processes in the computational grids. In *3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, 2015.

[4] e. a. Izakian, Hesam. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *International Joint Conference on Computational Sciences and Optimization*, 2009.

[5] K. Kc and K. Anyanwu. Scheduling hadoop jobs to meet deadlines. In *IEEE Second International Conference on Cloud Computing Technology and Science.*, 2010.

[6] C. E. Metz. Basic principles of roc analysis. In *Seminars in nuclear medicine. Vol. 8. No. 4. WB Saunders*, 1978.

[7] J. A. Reig, Gemma and J. Guitart. Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In *Ninth IEEE International Symposium on Network Computing and Applications*, 2010.

[8] B. Shahzad and M. T. Afzal. Optimized solution to shortest job first by eliminating the starvation. In *The 6th Jordanian Inr. Electrical and Electronics Eng. Conference (JIEEEC)*, 2006.

[9] S. K. S. R. N. M. Siddharth Dias, Sidharth Naik. A machine learning approach for improving process scheduling: A survey. In *International Journal of Computer Trends and Technology (IJCTT) V43(1):1-4*, 2017.

[10] V. T. Smith, Warren and I. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Workshop on Job scheduling strategies for Parallel Processing*, 1999.