Feature Selection for Learning to Predict Outcomes of Compute Cluster Jobs with Application to Decision Support

Okanlawon. Adedolapo[1], Yang. Huichen[1], Bose. Avishek[2], and Hsu. William[3], Andresen. Dan[4], and Tanash. Mohammed[4]
Department of Computer Science, Kansas State University
Manhattan, Kansas, USA

Abstract— *We present a machine learning framework and new test bed for data mining from the Slurm Workload Manager for high-performance computing clusters. The focus of this work is on feature selection methods for a decision support task: helping users decide whether to resubmit failed jobs with boosted CPU and memory allocations or migrate them to a compute cloud. We describe how this task can be cast as both a supervised classification and regression learning task and one of sequential problem solving suitable for reinforcement learning. Selection of relevant features can improve training accuracy, reduce training time, and produce a more human-comprehensible model, towards explainability of predictions and inferences made by the resulting intelligent system. We present a supervised learning model trained on a Slurm data set of HPC jobs using three different feature selection techniques: Linear Regression, Lasso, and Ridge Regression. Our data set was equally representative of both HPC jobs that failed and those that did not, thus making our model reliable, less likely to overfit, and being able to generalize. Our model achieved an R-squared of 95 percent and 99 percent accuracy. We identify five predictors for both CPU and memory properties such as CPUTimeRaw and MaxRSS.*

Keywords: HPC, predictive analytics, feature analysis, user modeling

1. Introduction

Information available at job submission time has been shown to help with prediction of resources required for jobs to run till completion. We extend our earlier work [1] by designing a model that is trained on a recent and improved Slurm data set collected from our high-performance computing (HPC) cluster at Kansas State University. Slurm is a widely used cluster management and job scheduling system for large and small clusters [2]. Our supervised learning task explores three different regression analysis methods namely linear regression, ridge regression, and lasso. Using these methods, we derived additional features that were not available at job submission time. There is a growing interest in leveraging machine learning in the estimation of resources in the past few years. Matsunaga et al. identified the best ML algorithms for predicting execution time and resource usage for jobs [3]. More recently, Mao et al, employ a variant of the policy gradient method, REINFORCE, to train a neural network to optimize for objectives such as minimizing average job slowdown and completion time [4].

Traditional approaches for scheduling jobs such as backfilling are largely based on workload heuristics which need a lot of domain knowledge, and are less adaptable to changing workloads. Practical application of scheduling improvements have been marred by failures due to emerging parameters during execution that were not initially considered. Our approach attempts to design methods that are robust and can generalize across platforms [5]. We believe exploring job characteristics can provide better decision support to the users and administrators of these systems when they specify resource estimates for their jobs. These characteristics can be collected at different times during the lifetime of a job including at submission time or when it is queued, during the job run, and after the completion or failure of the job. We also carried out a survey on active users of our system to capture any user behavior that could help model user behavior and determine the prospect of improving system users' habits. The combination of all these can better explain job scheduling patterns and help with design better scheduling techniques to improve optimization of system resources.

Our major contribution in this paper is the comprehensive feature analysis of job prediction factors using a representative data set collected from historical log files of compute clusters. These features were majorly related to the CPU and memory information of the jobs. Additional features are related to the time required for them to run on the cluster as well as the number of cores. We address the lack of adequate features available for performing accurate prediction of the resources required by these jobs. At submission time, users provide sparse information such as the number of cores needed or memory requested, which are insufficient for predicting the outcomes. Although there is a lot of postmortem data that can be gathered from historical log files after the job has been executed, this lack of predictive information at submission time significantly reduces the level of confidence with which we can predict job outcomes. Our results show a ranking of the most useful predictive features and we used these features for prediction of the entities important for both

cpu and memory.

Finally, we introduced ongoing work on gradient boosting decision tree framework based on LightGBM. This framework has advantages including faster training speed and higher efficiency, lower memory usage, better accuracy, support of parallel and GPU learning, and being capable of handling large-scale data. Our goal is to derive the best features for prediction using regression analysis techniques.

In the rest of the paper, we describe current and ongoing work, development of our data set, our design architecture and implementation, and evaluation of our results.

2. Machine Learning and Related Work

Our interest in this work is borne out of the recent growing interest in using machine learning techniques for optimization of HPC systems [3, 4]. While optimization of the HPC system is the end goal, we are focused on creating models that will help users with specifying accurate resource requirements for their jobs to run till completion, particularly those related to the CPU and memory. We believe making these models agnostic can help real-time monitoring of the state of the system and user requests across platforms [8]. Additionally, we intend to use features, historical data, and survey data collected to model users' behavior that can give better insights into their behaviors and possible ways to help encourage better scheduling habits and behaviors. Currently, users can only rely on heuristics and their past experience to help them with this process. While administrators managing these systems have monitoring dashboards to help with optimizing the system efficiency, it is impractical, especially for large clusters, for them to continually monitor individual jobs. This underscores the need for better analysis of users' specified resource estimates for their jobs.

2.1 Feature Analysis

One important part of a predictive modeling task is the selection of the most useful features for prediction. Feature selection is the process of constructing the subset of input features x, where x belongs to X to determine the output variable Y. By removing irrelevant and redundant features, it brings benefits such as improving accuracy, reducing overfitting, and saving computational time. We need to find the approximate relationship function f() between input X and output Y, but we usually do not need to take all the features available from the data set as inputs, especially when the data set contains hundreds or thousands of features. In general, there are three feature selection methods: filters, wrappers, and embedded methods [9].

Filter methods use statistical measures to assign a score for each feature. The features would be kept or removed from the data set in use by their ranking score. They are independent of the predictive model. These methods include chi-square test, correlation coefficient, and Fisher score.

Wrapper methods train a model with different combinations of subset of features, evaluate and compare the importance of features. The model will be iterated until the optimal subset is found[15]. Wrapper methods consider the selection as search problems which can be mainly divided into exhaustive search, heuristic search, and random search. Some common examples include forward selection, backward elimination, and recursive feature elimination.

Embedded methods integrate feature selection as part of the model training process. This combines the characteristics of filter and wrapper methods mostly geared towards reducing overfitting. The most common embedded feature selection method is regularization methods, such as Lasso and Ridge regression.

We used all of the three feature selection methods mentioned above as features for pro-processing in our experiment. Five different selector algorithms from the three methods are chosen: Random Forest for feature importance as filter feature selection method, Linear Regression, Lasso and Ridge Regression as embedded feature selection methods, and Linear Regression as wrapper selection method. All these methods are implemented using `scikit-learn` [10].

2.2 Gradient Boosting

Our feature selection focused on the prediction of two main features: MaxRSS and CPUTimeRaw. The former is the maximum resident set size of all tasks in a job while the latter is derived as the time used by a job or step in HH:MM:SS format. Table 1 shows a comprehensive list of all the features and their descriptions.

For our classification tasks, we explored decision-tree algorithms in conjunction with boosting methods such as Adaptive Boosting (AdaBoost) and LightGBM, a framework designed to make training faster and handle large amounts of data. With gradient boosting methods, we can train models in a gradual, additive, and sequential manner. Even though these sorts of methods are generally weak predictive models, they can be boosted in terms of performance while being able to train efficiently.

2.3 Graph Convolutional Networks

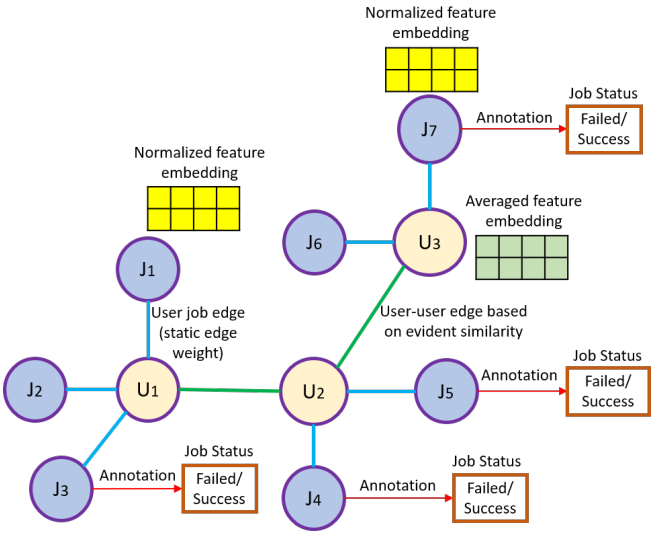
Recently, Graph Convolutional Networks (GCNs) [11] have drawn attention from the machine learning research community as a means of semi-supervised learning using fewer labeled nodes. In our application, this means bootstrapping the task of learning to predict outcomes for jobs from historical data about the submitting users or **related** jobs and users in a heterogeneous information network. Traditional deep learning model such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) consider each observation in the data sets is independent, and process the data set as a grid-based structure where the data set can be either text data set or image data set.

On the other hand, the GCN considers observations having inter-relation between them and works on graph structured data set or graph constructed from data sets. In fact, all grid based data set can be considered as a subset of graph structure data set but the vice versa of this relation is not valid most of the cases. In graph representations of system data, each observation is represented as a node, and the relationships between the observations are represented by edges that connect nodes together. Being a very new approach, GCNs show outstanding performance in terms of accuracy, precision and recall, and in many use cases outperforms in both binary and multiclass classification analyses even if the data set is not sufficiently annotated compared to its CNN and RNN counterparts. A key challenge, however, is to determine the good metrics to make relations between the distinct observations in our Slurm data set.

2.4 GCN in HPC Analytics

To address the challenge of ensuring reliable computing among HPC-cluster users, we should consider that users have prior knowledge of using the HPC computing resources. As previously mentioned, we often lack the privilege of having sufficiently experienced computing cluster users. This inexperience can often lead to inaccurate estimation of resource needs and requests by unskilled users. Job management platforms such as Slurm does not require sufficient information directly from user while getting a job request from a user. As we have scarsity of user specific information in the Slurm data set, we have to do both exploring probable and extracting implicit relation in existing user information from the data set. Interestingly, if we use the optimally utilized user information from the Slurm data set, that can be a leap to predict user future job status, and to recommend required resources to the user before the job being submitted. Although there has been a good amount of research works on Graph Convolutional Network (GCN) conducted on different domains such as social network analysis [13], natural language processing [14], signal processing, etc., to the best of our knowledge it will be the first work of applying GCN on HPC data. As we mentioned earlier, the challenge is to find relations between the observation in the HPC data set. As the Slurm data set includes some user information and we have collected some user survey data while the users submit their jobs, we can make relations between the observation or datarows from the data set. Surprisingly, in the Slurm data set, we do not have the limitation of having annotated data because each row in the data set represents a job submission event, and every row has an outcome whether the job failed or not. Our current goal is to create a graph from the Slurm data set and train a GCN model on this graph data set.

Fig. 1: Graph Structure on HPC data for GCN



2.5 Methodology of Applying GCNs to HPC data

Figure 1 illustrates the modular structure of the user-job heterogeneous information network. User nodes and job nodes label names start with 'U' start with 'J' respectively which are shown in different color. User nodes are connected through edges that are derived from the relations we will extract from the data set. Job nodes are not connected directly but they are connected through user nodes. We will use the status of submitted job as the target for training the GCN model whether the job is failed or succeed. We will extract and then preprocess some features from the Slurm data set. After that we will normalize the numeric features using min-max normalization and make an embedding from the datarow for each job submitted. The user node embedding will be calculated by averaging all the embeddings of corresponding jobs that this user submitted. Finally, we can train a two layer GCN model on this graph data set. This approach is still in the implementation phase and part of continuing work described in the Current and Future Work section.

2.6 Stay-or-Go

In addition to insights derived from analyzing job-related and user-related information, we emphasize the importance of scheduling decisions made after jobs have been submitted and failed. There is a correlation between the failure of HPC systems and the type and intensity of the workload running on it. Thus, we believe decisions made by users when they readjust their job parameters during resubmissions are important. As a result of this, we explore the task of evaluating the utility obtained when job runs on the local computing cluster for both one-time submissions and resubmissions after job failures. We comparatively evaluate this

utility as well as that derived from migrating the job instead to a compute cloud such as Amazon Web Services (AWS) [13] and the Google Cloud Platform (GCP) [14]. With this information, users can make more informed decisions not just on how to set their job parameters, but also to decide what platform to run their jobs and at what cost.

Even though CPU and memory characteristics are largely important for predictive analytics of HPC jobs, we are interested in the cost incurred by a job when it occupies a number of cores on the system. One way we define this cost is in terms of the length of time it takes for the job to run to completion if it was a one-time submission. For resubmitted jobs, this would be the time expended for each past submission up this point, as well as expected time it would require for this job to finish on the cluster. Another dimension is the dollar cost of running the job. For most on-premise cluster as in our study, the dollar cost is defined as a flat cost even though the system is made available for free as for a large number of our users. We can define a relationship between this on-premise cluster cost and the cost of migrating to cloud which can help users make the best choices.

Several studies have analyzed the performance of HPC applications on the cloud with interesting results[6, 7]. These studies have demonstrated the viability of the cloud for HPC applications, though with some concerns[8]. Some of the concerns raised when cloud is adopted as an alternative to running HPC jobs include trade-offs in performance, interoperability, and security issues.

2.7 Potential for Reinforcement learning

We believe that reinforcement learning techniques can be adopted to the Stay-or-Go task of helping users decide when to stay on the local cluster or 'cloudburst'. By cloudburst, we mean the migration of the job to a cloud cluster especially during peak periods [17]. We model this as a sequential decision-making problem where an agent can be specified as a partially-observable Markov Decision Process (poMDP) [16]. Through this, we can capture the uncertainty in decision-making especially apriori decisions made during multiple submissions of a job that continuously fails. This will serve as a basis to optimize the cost/utility function given below, helping users make the decision of resubmitting their jobs and what platform to do so.

$$\sum_i C_i * P[(i, killed)|params_i] + C_A$$

where

$$C_i$$

is the cost of all runs up to a time t, while

$$C_A$$

is the extra cost of stay-or-go. As mentioned earlier, this is evaluated in terms of time spent on either or both platform and monetary cost of acquiring the cores to run the jobs.

Table 1: Features of the original data set

Account	JobName	AllocCPUS
AllocNodes	AveCPU	AveCPUFreq
AveRSS	CPUTimeRAW	ElapsedRaw
Eligible	End	JobID
MaxDiskReadTask	MaxDiskWrite	MaxDiskWriteTask
MaxRSSTask	MaxVMSize	MaxVMSizeTask
NCPUS	NNodes	NTasks
ReqCPUS	ReqMem	ReqNodes
State	Submit	SystemCPU
Timelimit	TotalCPU	UserCPU

3. Experiment Design

In this section, we describe the acquisition and preparation of training data for machine learning, the principles governing our feature extraction and selection process, and design choices for learning algorithms and their parameters.

3.1 Data Preparation

The data set has been collected from Simple Linux Utility for Resource Management (Slurm) database. It represents the user job submission history from the year 2018 to 2019 of the primary Beocat HPC system and has 10.9 million instances. The raw Slurm data set has 105 features which record the details of parameters of any job that has finished running, such as the number of resources that user required when submitting the job, the number of resources that system allocated to the job. However, the raw data set has duplicated features, and some features have missing values. We clean the data set based on the following rules:

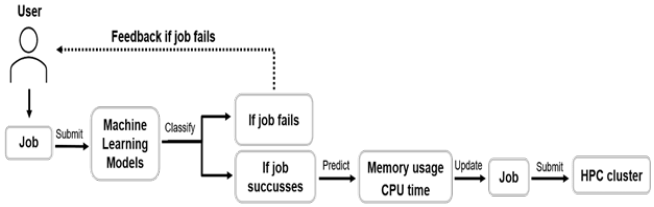
- Remove meaningless features from the data set. For example, some features only have two values that are either very close to each other or one of them is unknown. The lack of variation in the features leads to unsuccessful regression results
- Remove the features which have NaN values
- Three Remove the jobs with running state
- Change all of the feature values to numeric. For instance, change normal time format (Year-Month-Day, HH:MM:SS) to Linux timestamp

3.2 Design Architecture

In this proposed approach both user and job will be represented as nodes. User-user relation will be made by considering various evident similarity metrics such as i) the project they are working, ii) the department they are from, iii) types of their submitted jobs, iv) job submission time, v) similarity of requesting resources, etc. We will select only those features from the data set which are important for class prediction by using feature selection algorithm.

As the features' numeric values cover a wide range, we will do min-max normalizaiton to make normalized feature embedding for job nodes. User node feature embedding will be calculated by averaging all submitted jobs by the user. We will split the data test and train set to train the model and predict the outcome whether the job will fail due to insufficient resources.

Fig. 2: resource estimation pipeline



3.3 Machine Learning Implementation

To help improve prediction, we preprocessed our data set using the `scikit-learn` [10] Python library before running our experiments, as documented in this section.

Our selected predicted features for CPU and memory were CPUTimeRaw and MaxRSS for CPU and memory respectively. We created feature ranking scores that shows the most useful features for prediction for three different methods: Linear Regression, Lasso, and Ridge Regression. We then aggregated these scores by calculating a mean score for our rankings. The results for our top 5 features are shown in Tables 2 and 3.

Table 2: Feature ranking scores for best features for predicting CPUTimeRaw

Features	ranking score
ElapsedRaw	0.72
AllocCPUS	0.66
NCPUS	0.39
ReqCPUS	0.36
Submit	0.35

Table 3: Feature ranking scores for best features for predicting MaxRSS

Features	ranking score
AveRSS	0.76
NNodes	0.62
AllocNodes	0.61
ReqMem	0.54
Submit	0.48

4. Evaluation

In this section, we discussed the ranking of the predictive features using the mean score we discussed earlier. This was derived by the mean of the performance of all features across the three feature selection techniques. We also showed the performance of these predictors by applying linear regression on the data set and evaluating the predictive strength. This was evaluated metrics including accuracy, F1, and AUC as metrics.

Tables 2 and 3 show the best five features among the original 105 features in the data set for prediction. This was calculated by using the mean scores. We observe that only AllocCPUS is highly predictive for the CPUTimeRaw by 70 percent while AveRSS and number of nodes (NNodes) perform better for MaxRSS. As expected, Table 4 shows the worst performing predictors for CPUTimeRaw are mainly features related to the memory with little or no predictive abilities. Similarly, this is observed for MaxRSS as shown in Table 5.

Table 4: Feature ranking scores for worst features for predicting CPUTimeRaw

Features	ranking score
MaxRSS	0.00
AveRSS	0.01
AveCPUFreq	0.02
NTasks	0.03
MaxRSSTask	0.04

Table 5: Feature ranking scores for worst features for predicting MaxRSS

Features	ranking score
MaxRSS	0.00
AveRSS	0.01
AveCPUFreq	0.02
NTasks	0.03
MaxRSSTask	0.04

The next step was to run a linear regression model using these features selected to predict the MaxRSS and CPUTimeRaw. We also used k-fold cross validation with k = 5 selected empirically to reduce noise due to noisy data. We adopted the accuracy score library provided by scikit-learn to evaluate our models. The accuracy is the number of correct predictions made by our model divided by the total number of predictions made. F1 scores weighs both the precision and the recall, where an F1 score reaches its best value at 1 and worst at 0. Finally, R-squared measures the percentage of the variance explained by our models, that is the fraction by which the variance of the errors is less than variance of our predictors. Tables 5 and 6 show the results for both MaxRSS and CPUTime based on the metrics explained earlier.

5. Conclusions and Future Work

In this paper, we implemented a supervised learning model to extract the most useful features for prediction in HPC resource estimation on a Slurm computing cluster. Our model

Table 6: Classification results for MaxRSS

Model	Accuracy(%)	F1(%)	R squared (%)
LR	93	95	95
LR with k-fold	98	96	95

Table 7: Classification results for CPUTimeRaw

Model	Accuracy(%)	F1(%)	R squared (%)
LR	96	91	93
LR with k-fold	99	92	90

achieved 99 percent accuracy and confirmed the predictive ability of our model. Our model showed that there are a few that are good predictors for scheduling that can be further explored.

References

[1] Andresen, D., Hsu, W., Yang, H., Okanlawon, A. : Machine learning for predictive analytics of compute cluster jobs in *The 16th International Conference on Scientific Computing*.

[2] Yoo, Andy B., Morris A. Jette, and Mark Grondona. "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 44-60. Springer, Berlin, Heidelberg, 2003.

[3] Matsunaga, Andréa and Fortes, José AB. "On the use of machine learning to predict the time and resources consumed by applications," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 495-504. IEEE Computer Society, 2010.

[4] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. 2016. Resource Management with Deep Reinforcement Learning. In HotNets. ACM.

[5] Jérôme Lelong, Valentin Reis, and Denis Trystram. 2017. Tuning EASY-Backfilling Queues. In 21st Workshop on Job Scheduling Strategies for Parallel Processing.

[6] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 395–401

[7] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, and G. C. Fox, High Performance Parallel Computing with Clouds and Cloud Technologies. CRC Press (Taylor and Francis), 07/2010 2010.

[8] Abhishek Gupta, Paolo Faraboschi, Filippo Gioachin, Laxmikant V. Kale, Richard Kaufmann, Bu-Sung Lee, Verdi March, Dejan Milojicic, and Chun Hui Suen. 2016. Evaluating and improving the performance and scheduling of HPC applications in cloud. IEEE Trans. Cloud Comput. 4, 3 (2016), 307–321

[9] Guyon, I., Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of machine learning research, 3(Mar), 1157-1182.

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830

[11] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." in *arXiv preprint arXiv:1609.02907* (2016).

[12] Yao, Liang, Chengsheng Mao, and Yuan Luo. "Graph convolutional networks for text classification." in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 7370-7377. 2019.

[13] Pei, Hongbin, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. "Geom-gcn: Geometric graph convolutional networks." *arXiv preprint arXiv:2002.05287*, (2020).

[14] Yao, Liang, Chengsheng Mao, and Yuan Luo. "Huckman R., Pisano G., Kind L. (2012), 'Amazon web services,' Harvard Business School case number 9-609-048. Harvard Business School Publishing: Boston, MA.

[15] R. Kohavi, G.H. John Wrappers for feature subset selection Artif. Intell., 97 (1997), pp. 273-324

[16] Rao, R. P. N. Decision making under uncertainty: a neural model based on partially observable Markov decision processes. Front. Comput. Neurosci. 4, 146 (2010).

[17] S.K. Nair, S. Porwal, T. Dimitrakos, A.J. Ferrer, J. Tordsson, T. Sharif, et al., "Towards Secure Cloud Bursting Brokerage and Aggregation", Proc. Eighth IEEE European Conf. Web Services (ECOWS '10), pp. 189-196, 2010.