Hybrid Prediction models for CPU Burst-Time Prediction

1st Gurpreet Singh
Graduate Engineering
Santa Clara University
Santa Clara, CA
gsingh8@scu.edu

2nd Rohan Pandey
Graduate Engineering
Santa Clara University
Santa Clara, CA
rpandey4@scu.edu

3rd Sudhanva Suresh Aithal
Graduate Engineering
Santa Clara University
Santa Clara, CA
ssureshaithal@scu.edu

***Abstract*—Utilizing CPU optimally is pertinent for making the processes execution look parallel. Some of the CPU scheduling algorithms like Shortest-Remaining-Time-First (SRTF) and Shortest-Job-First (SJF) need CPU burst time, beforehand so that the processes in the wait queue can be scheduled in an optimal way. There are various existing mathematical techniques to find out the burst time of the process but they are proved to be expensive. This paper proposed several hybrid prediction machine learning models and feature selection techniques are used. ML model includes Linear Regressor, Ridge Regression, Lasso, Random Forest (RF) Regressor, Support Vector Regressor (SVR), K-Nearest Neighbor (KNN) and Decision-Tree Regressor (DT) have been used. Models have been trained using GWA-T-4 AuverGrid dataset provided by TuDelft. Mean Absolute Error and R2 Score are used to evaluate the trained algorithm's performance. On testing data, the Decision-Tree and Random Forest outperformed every other regression algorithm which achieves an R2 of 0.90, while K-nearest, SVR, and Linear Regression achieve 0.8754, 0.8045, and 0.7926, respectively. Empirical evidence demonstrates the superiority of specific attributes over others for improved outcomes**

***Index Terms*—CPU Task Scheduling, CPU Burst Time, ML Algorithms, Operating System**

I. Introduction

To make processes execution look parallel or in other words, to enable multi programming there is need to schedule CPU in such a way that make the maximum CPU utilization. There are some algorithms, first one Shortest Job First and secondly Shortest Remaining Time First which push process to the ready-queue based on the burst time. These two algorithms schedule the processes having small burst length that means processes with small burst will have high chances of getting CPU and will release the CPU quickly and won't hold to it. Burst time is simply a time between when a process is assigned to the CPU till it is taken back from it, this time period is usually refer to as burst time [2]. These algorithms executes based on burst time, these algorithms need to know the burst time for each process, beforehand, so that they can schedule based on these values. Predicting CPU Burst Time is a challenging tasks, there are various mathematical formulas and ways to calculate the CPU burst time but these methods may sometime doesn't provide satisfactory values and hard to calculate. Trained machine learning model will be used as substitute in calculating CPU bust time. In following sections methodology is discussed (Section II). Sections III and Sections IV will discuss proposed machine learning approach and results and discussions. Various machine learning models have been broadly utilized in the various domains – healthcare, military, medical science, Weather forecasting [6],[7],[8],[9]. Similarly, it can be utilized to predict burst time as well. This paper has applied several machine learning algorithms, Linear Regression, Ridge Regression, KNN, Random Forest Regression, Decision-Tree Regression, Support vector Regressor to predict CPU Burst time. Along with the machine learning models we tried to find out that is there any relation with the reduced features and the model efficiency because it is possible that some attributes as a whole degrades the performance of the model. Although these selection techniques do not guarantee to augment performance, we have used two feature selection techniques to check the possibility. Executing model training is not only providing the data set to train the model but to clean and pre-process the dataset before tuning according to the model. We did inferencing and clearning on the dataset before using for model training and evaluation. The performance of the hybrid regression models are evaluated on the basis of metrics – R2 score. Higher the R2 score, higher is the chance of the model being able to explain the variance in the predicted output. Moreover, Model performance will be evaluated on the following cases: un-optimzied model, reduced feature model and finally based on hyper-parameterized machine learning model.

II. METHODOLOGY

A. Support Vector Regresssor

This ML algorithm which is used for predicting continuous value. It is applied to regression and classification issues. Each yi coordinate is labeled with either zi = 1, 1, or a diagonal matrix Y containing zi. A hyperplane f differentiates across classes (p, q). The data samples are shown as

$$y_i.p + q >= 1 \quad z_i = +1 \tag{1}$$

$$y_i.p + q - 1 \quad z_i = -1 \tag{2}$$

SVR obtain the ideal plane so as to maximize the distances between nearby data points. The ideal hyperplane equation is given as

$$y_i.p + q = 0 \quad p^n, \quad q \in R \tag{3}$$

Here the term u is a normalized vector to hyperplane f(p, q) and the character v is a known bias.

$$Minimize 1/2|k|^2 + C[\sum_{n=1}^{m} \xi_n \in n = 1] \tag{4}$$

B. Adaptive Boosting

Freund and Schapire [22] presented the adaptive boosting (Adaboost) technique, formerly referred to as AdaBoost.M1. It was the first practical and extensively implemented boosting method for binary classification. By weighing the observations, the primary purpose of this approach is to reduce the error generated. It operates by allocating greater weights to incorrectly categorized samples and smaller weights to successfully classified samples. This approach is learned sequentially by linking several weak learners, with errors minimized by adjusting the observation and classifier weights. For the better result on the testing data, this algorithm continues to work until it discovers the minimized error. The most probable single strong prediction. Let's examine the mathematical portion. Let n is the total number of dataset and y is the respective label $(y_1, z_1) to (y_n, z_n$, where y_n belongs to X.

III. MODELS PROPOSED

We have proposed 6 hybrid models for CPU prediction. The base idea is to use a combination of two feature selection techniques which is Gradient boosting (GB) and second one is Random Forest (RF) with 6 types of predictive machine learning models. To get an optimal predictive model it is very pertinent to select input variables and important features carefully, as the classifier performance can be degenerated by irrelevant or redundant variables. Gradient boosting and Random Forest feature selection techniques are the two similar techniques that will help us in calculating the importance of the feature. There are two main intention of selecting features. First, it will help to reduce the over-fitting of the model as well as to find out what are the important CPU's attributes that have higher contribution in CPU burst time prediction. RF has the advantage that it does not over-fit easily with the number of trees, the same was demonstrated by Leo Breiman [23]. Selecting models is not an easy choice for carrying out the project because it take long hours to train a model hence, it is mandatory to select the model according to the end goal as well as based on the present data. In this project, goal is to predict the continuous value and for that we have to choose supervised machine learning algorithms that can predict continuous value of data. Once the model is selected and trained (un-optimized) we will try to train base on the optimal feature selected. In final stage we will tune the hyper-parameters that generally don't get trained during model training. Optimal model will be selected form the previous stage and will be used for hyper-parameter tuning. In our case it is decision Tree which will be used for hyper-parameter tuning which is based on several parameters let's say, splitter, max depth, max leaf nodes etc. This project uses gwat4auvergrid dataset provided by TuDelft. This dataset contains data from around 0.45 million records along with 29 distinct features. The classifier data is divided into an 80-20 ratio to facilitate the testing and training of the hybrid prediction model. Below is the amount of data we have used as training date and testing data.

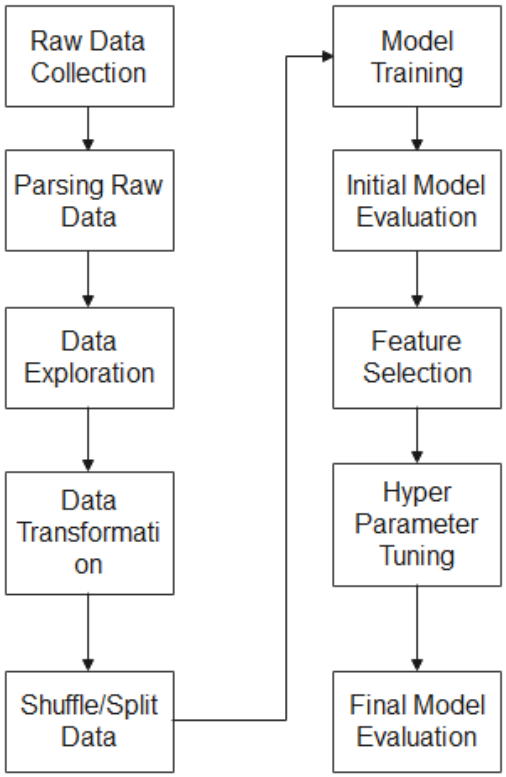

Fig. 1. Research Project Flow

TABLE I
SPLITTING DATASET

Division Type	*Training*	*Testing*
Number of Records	278088	69523

A. Data Collection/Parsing/Exploration/Transformation

This project uses gwat4auvergrid dataset provided by TuDelft. This dataset contains data from around 0.45 million

records along with 29 distinct features of CPU processes and other attributes. Initially data was in Raw format hence it is pertinent to transform into csv file which is used for model training. A separate python script is created for parsing raw data into CSV file. For data transformation, we need to have detailed view and the understanding of the data.

1) Removing Unavailable data: It is noticed that a few of the attributes had more than 80% rows values missing (represented by -1). If used for training it will degrade the performance, hence we have used threshold value of 80% which will decide to drop the column or not. In this phase we have dropped 12 columns - ReqMemory, UsedNetwork, UsedResources, ProjectId etc

2) Removing Skewness: There were several columns in which data is skewed to a particular range. This kind of data will make our models bias to certain values which is not suitable for prediction. Hence, we have used log-transformation to remove the skewness and to keep the data value in a particular range. Figures below show the skewness and when skewness is removed.

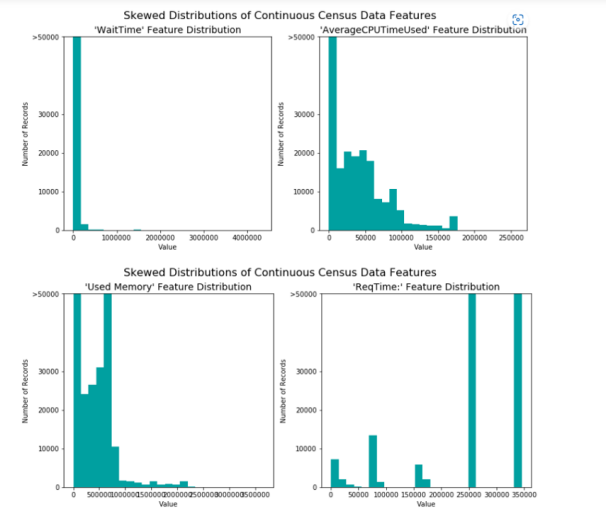

Fig. 2. Skewed Data

B. Shuffle and Splitting Data/ Model Training/ Hyper-Parameter Tuning

After all the transformation is done, next step is to shuffle and split the data into training and testing set for model training an testing. Initially we have 0.45 million of records available and 29 features. After transformation we are left with around 28000 number of records. All 6 models are trained on the transformed data and stored the results.

1) Feature Selection: In this phase we tried to check if certain features are enough to predict the CPU burst time. We have used two feature selection techniques which will produce importance feature vector based on the data we provided as input, We plotted importance vector as shown in Figure 6 and Figure 7. Predictive models again got trained with the lessen

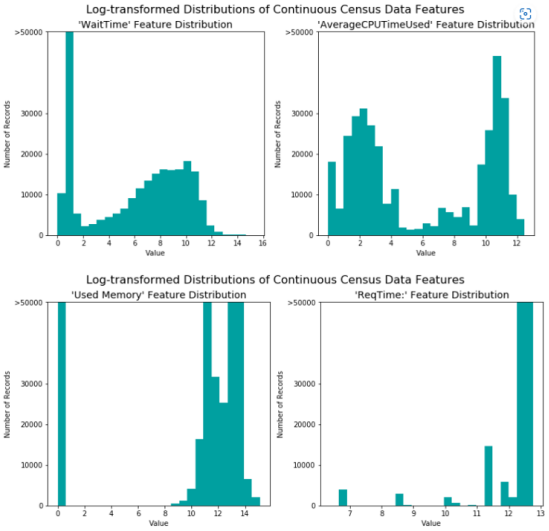

Fig. 3. Normalized Data

and important values. After CPU burst time prediction out of 17 relevant features, 14 features were given high importance. Trained Model results shown in Figure III.

2) Hyper-Parameter Tuning: There are several approaches to tune the hyper-parameter in this project we have used 'K Folds Cross Validation' technique [24]. The data used for training is trimmed into K-continuous batches of similar size. The reason for splitting into various block is to use One block for validation and the rest k-1 block will be used as training dataset. The process take a lot of time to tune the parameters and it will get terminated when there is no batch is left for validation. To calculate the result, average of the validation of each batch is used. The above-mentioned process will be performed the same way for each of the hyper-parameter values provided and we will store the parameter which will provide high validation accuracy.

C. Measure Of Performance

We will use two metrics to evaluate the performance of machine learning models i.e. Mean Absolute Error and R2 Score. MAE or Mean absolute error is a simple model evaluation metric which calculates the mean of sum of absolute difference between predicted and actual values. It is calculated using the below-mentioned equation.

$$MAE = 1/N + \sum_{n=1}^{r} |y - y^{'}| \tag{5}$$

Here r is the number of records. The above-mentioned MAE equation helps in summing all the errors across all the records used for training and testing. It is expected to have minimum MAE because it is kind of a loss function, hence having minimum errors across observations means better model.

There is another metric which is used to evaluate model's performance, R2 score. This metric explains how well a model has performed on the training and testing data and doesn't

define the error as earlier defined by Mean Absolute Error. Higher R2 score signifies better model performance. It is calculated using the following mathematical equation

$$R2Score \ = 1 - SSr/SSm \tag{6}$$

SSr = sum of error over regression line,squared
SSm = Square of sum of error over mean line

IV. RESULTS AND DISCUSSION

We have already discussed the calculation of performance for predictive algorithms by the metrics R2-score. The output recorded in Table II and Table III , clearly concludes the best performance of Decision Tree and Random Forest which has highest R2-Score (shown in Blue) compared to other predictive classifiers in the initial model evaluation. Among these two regression models, decision Tree performed well and hence selected for further hyper-parameter tuning. Results showed that tuning hyper parameters for Decision Tree increase the performance significantly, from 0.75 R2 score to 0.90 R2 Score. In the given dataset we had around 29 features, after dropping irrelevant features we are left with only 17. In thought of increasing model performance we have applied Random Forest and gradient boosting to the selected predictive models. The feature selection techniques in-fact decreased the model performance. Decrease in performance is justified by the fact that decrease in features effects models to get over-fitted on the similar dataset. Hence it would be better increasing the feature set and that might help in increase the performance of the model. We observed that majority of contribution in CPU prediction is done by not more than 17 features out of a total of twenty-nine features. The performance of Hyper-parameterized tuned Decision Tree models is better than other models. We can conclude that feature selection might not help in really augment the R2 Score but tuning hyper-parameters can significantly increase the performance and reduce the computational time.

TABLE II
MODEL PERFORMANCE RESULT 1-3 MODELS

Performance Metric	**Performance Evaluation**		
	Linear Regression	***Lasso***	***Ridge***
R2 Score	0.62	0.58	0.62

TABLE III
MODEL PERFORMANCE RESULT 4-6 MODELS

Performance Metric	**Performance Evaluation**		
	KNN	***Random Forest***	***Decision Tree***
R2 Score	0.45	0.75	0.78

Final Result Analysis			
Metric	Unoptimized	Reduced Feature	Optimized
R2 Score	0.7816	0.8284	0.90

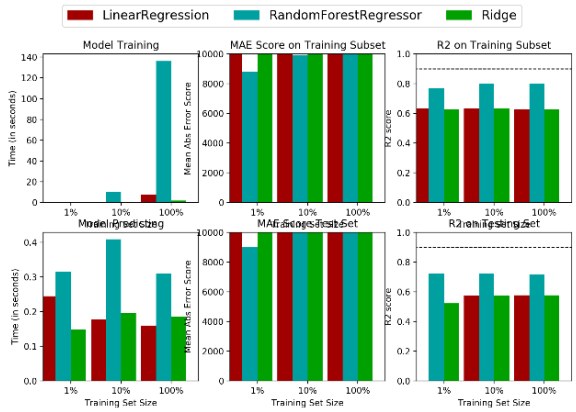

Fig. 4. Model Performance Result on First 3 Regression Models

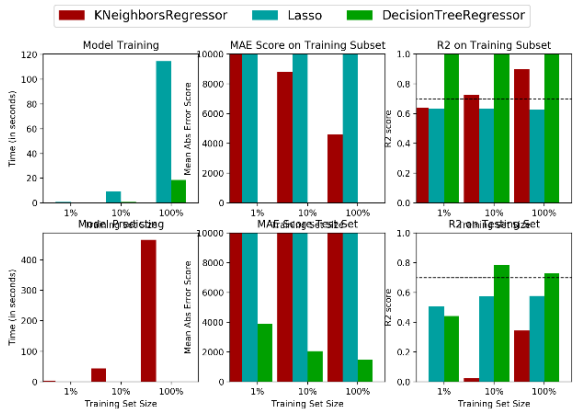

Fig. 5. Model Performance Result on last 3 Regression Models

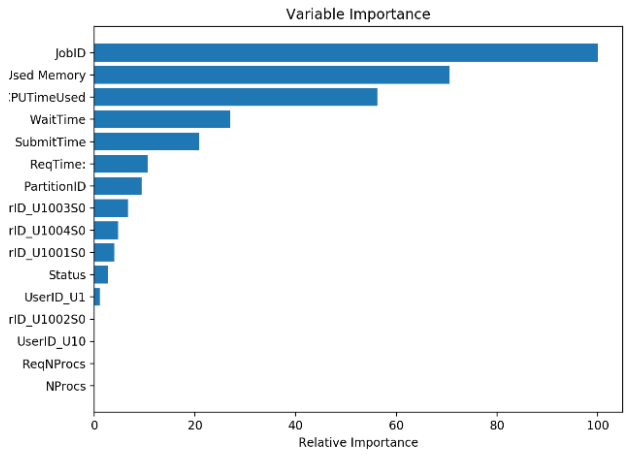

Fig. 6. Feature importance using Random Forest.

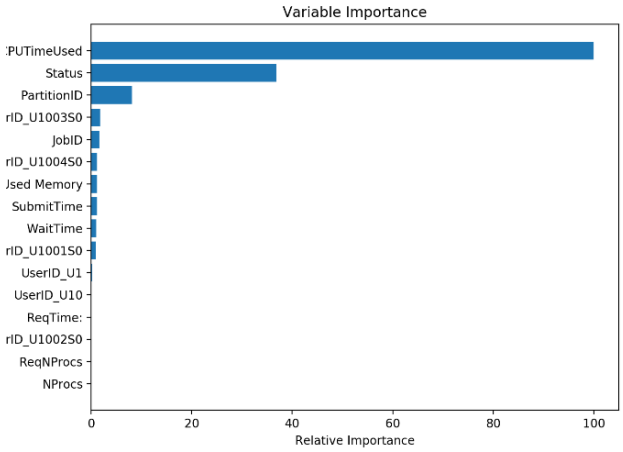

Fig. 7. Feature importance using Gradient Boosting

V. Conclusion

In this paper, we have come up with 6 machine learning regression models - Random Forest, Linear Regression, Ridge Regression, Decision Tree and KNN. We have used various approaches for data cleaning and data transformation which is used for initial model training. We have used R2 score to evaluate the model performance, it describe how well our model is able to explain the various of the output. Feature selection techniques are used to select best parameters out of 29 attributes listed which resulted into importance feature vector, we have trained out model again on first 10-14 important features. This reduction in feature set resulted in decrease in R2 score. There are various features of the models which doesn't get trained during the training phase hence, we have used Cross validation for tuning these parameters. Decision Tree is used as perfect model for hyper-parameter tuning because it had performed better in initial phase. Decision Tree is trained on several parameters and optimal parameter is used for further model training. This phase lead to increase in the model performance significantly from 0.82 to 0.90 (Table III).

VI. Future Work

This research project has mainly focused on the feature selection and training model on limited set of features and tried to evaluate the trained model performance. In the future work, new feature set can be explored where instead of reducing the feature set using feature selection, we can try to augment the feature set. Reason behind increasing the feature set is to reduce the overfitting of the model as number of records are much larger than the number of features. In the future scope, more feature generation could help in increasing the performance more than 0.9. Secondly, in future an end-to-end working model of scheduling algorithm can be implemented where each process will get burst time on runtime based on the pickle file (trained model backend). This way we can check the performance of CPU in real world instead of depending on some performance evaluation metric.

References

[1] T. Helmy, S. Al-Azani and O. Bin-Obaidellah, "A Machine Learning-Based Approach to Estimate the CPU-Burst Time for Processes in the Computational Grids," 2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS), 2015, pp. 3-8, doi: 10.1109/AIMS.2015.11.

[2] Schmitz, G. H., and J. Cullmann. "PAI-OFF: A new proposal for online flood forecasting in flash flood prone catchments." Journal of hydrology 360.1-4 (2008): 1-14.

[3] Riordan, Denis, and Bjarne K. Hansen. "A fuzzy case- based system for weather prediction." Engineering Intelligent Systems for Electrical Engineering and Communications 10.3 (2002): 139-146.

[4] Guhathakurta, P. "Long-range monsoon rainfall prediction of 2005 for the districts and sub-division Kerala with artificial neural network." Current Science 90.6 (2006): 773-779.

[5] Pilgrim, D. H., T. G. Chapman, and D. G. Doran. "Problems of rainfall-runoff modelling in arid and semiarid regions." Hydrological Sciences Journal 33.4 (1988): 379-400.

[6] Lee, Sunyoung, Sungzoon Cho, and Patrick M. Wong. "Rainfall prediction using artificial neural networks." journal of geographic information and Decision Analysis 2.2 (1998): 233-242.

[7] French, Mark N., Witold F. Krajewski, and Robert R. Cuykendall. "Rainfall forecasting in space and time using a neural network." Journal of hydrology 137.1-4 (1992): 1-31.

[8] Charaniya, Nizar Ali, and Sanjay V. Dudul. "Committee of artificial neural networks for monthly rainfall prediction using wavelet transform." Business, Engineering and Industrial Applications (ICBEIA), 2011 International Conference on. IEEE, 2011.

[9] Noone, David, and Harvey Stern. "Verification of rainfall forecasts from the Australian Bureau of Meteorology's Global Assimilation and Prognosis(GASP) system." Australian Meteorological Magazine 44.4 (1995): 275-286.

[10] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[11] Haykin, Simon. Neural networks: a comprehensive foundation. Prentice Hall PTR, 1994.

[12] Rajeevan, M., Pulak Guhathakurta, and V. Thapliyal. "New models for long-range forecasts of summer monsoon rainfall over North West and Peninsular India." Meteorology and Atmospheric Physics 73.3-4 (2000): 211-225.

[13] Luk, K. C., James E. Ball, and Ashish Sharma. "A study of optimal model lag and spatial inputs to the artificial neural network for rainfall forecasting." Journal of Hydrology 227.1-4 (2000): 56-65.

[14] French, Mark N., Witold F. Krajewski, and Robert R. Cuykendall. "Rainfall forecasting in space and time using a neural network." Journal of hydrology 137.1-4 (1992): 1-31.

[15] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995), pp. 273-297.

[16] Cao, Li-Juan, and Francis Eng Hock Tay. "Support vector machine with adaptive parameters in financial time series forecasting." IEEE Transactions on neural networks 14.6 (2003): 1506-1518.

[17] Kim, Kyoung-jae. "Financial time series forecasting using support vector machines. "Neurocomputing 55.1-2 (2003): 307-319.

[18] Radhika, Y., and M. Shashi. "Atmospheric temperature prediction using support vector machines." International Journal of Computer Theory and Engineering 1.1 (2009): 55.

[19] Vapnik, Vladimir. Statistical learning theory. 1998. Vol. 3. Wiley, New York, 1998.

[20] Vapnik, V. "The nature of statistical learning theory Springer New York Google Scholar." (1995).

[21] Vapnik, Vladimir Naumovich. "An overview of statistical learning theory." IEEE transactions on neural networks 10.5 (1999): 988-999.

[22] Freund, Yoav, and Robert E. Schapire. "A decision- theoretic generalization of on-line learning and an application to boosting." Journal of computer and system sciences 55.1 (1997): 119-139.

[23] Criminisi, Antonio, Jamie Shotton, and Ender Konukoglu. "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning." Foundations and Trends in Computer Graphics and Vision 7.23 (2012): 81-227.

[24] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. “Unsupervised learning.” The elements of statistical learning. Springer, New York, NY, 2009. 485-585.