Computational Grid CPU Burst Time Estimation Using Machine Learning Algorithms

Ashfaq Ali Shafin
FIU
Miami, Florida 33199
ashaf016@fiu.edu

Aseem Sharma
FIU
Miami, Florida 33199
ashar097@fiu.edu

Kratin Tiwari
FIU
Miami, Florida 33199
ktiwa003@fiu.edu

***Abstract*—CPU scheduling methods like Shortest-Job-First (SJF) and Shortest Remaining Time First (SRTF) relies on determining the burst time for the ready queue to operate. There are numerous approaches for predicting the duration of CPU bursts, such as exponential smoothing, although these strategies may not provide precise or consistent results. We propose a Machine Learning (ML) based solution to anticipate the duration of CPU bursts for processes in our study that extends the previous work of Helmy et al. [1]. Using a grid workload dataset, seven different machine learning approaches are employed to test and assess the suggested strategy. The results of the experiments reveal that the process characteristics and CPU Burst time have a strong linear connection. Furthermore, in terms of Coefficient of Determination (CD), Random Forest outperforms practically all techniques. In the case of the K-NN classifier, feature normalization enhances the outcome significantly.**

***Index Terms*—CPU Burst Time, Machine Learning, Feature Normalization**

I. Introduction

Process scheduling is considered to be one of the fundamental duties of any operating system. In execution mode, a program is considered as a process and in this era of multi-tasking, multiple processes are needed to execute concurrently. The process scheduler's task is to proliferate system performance in line with the set of criteria specified. The process shifts from being in the ready state to the running state. The work done by the CPU scheduler is to select one of the processes that are sitting in the ready state waiting to be executed and the CPU is assigned to one of them. To name a few process scheduling algorithms: Priority Scheduling, Shortest Remaining Time First (SRTF), Shortest Job First (SJF), Round-Robin (RR), First Come First Serve (FCFS), and so on [2].

The time spent while executing a process on the CPU from the point when it was scheduled to when it finally goes to sleep or exit after being in the running state is known as CPU Burst Time [3]. Many process scheduling algorithms like Shortest Remaining Time First (SRTF), and Shortest Job First (SJF) require the former information of process burst time before execution [2]. Process scheduling algorithms like SJF support short processes instead of long processes such that the process with shorter burst times completes and exits the system as soon as feasible, making it available for processes with longer burst times. Eventually, this helps reduce the number of processes in the waiting queue. Furthermore, it reduces the number of processes that are queued up behind the long processes. [4]. Different modified and improved versions of the round-robin algorithms also need to know the largest and smallest process burst time so that the time quantum or time slice can be modified dynamically instead of using static time quantum.

A group of mutually dependent supercomputers in a collection is called Grid computing. They usually work as a virtual computers to work on huge tasks such as data analysis and weather forecasting. With the help of the cloud the users can develop and implement large computer grids for work suited to their needs, this means that they only have to pay for what they use. This keeps the costs down as it helps with the deployment and construction of resources instead of doing it themselves, furthermore, it also saves time as it is quicker. As jobs are divided among several computers, there is a decrease in the processing time, also, the efficiency is increased and there is overall waste is decreased [5]. There are 3 phases as stated by [6], algorithms are important during the first stage, the design stage because they can assist in the discovery of a configuration capable of achieving a specific performance level. Secondly, the operational stage where allocation and assignment of jobs occur, and lastly, the reconfiguration stage where jobs are executed and tasks are reassigned. An additional dedicated machine in the grid is responsible for distributing and assigning the resources to the processes, this is done in the second phase. Therefore, the aforementioned dedicated machine should be able to make a prediction on the next CPU burst time for the scheduling algorithms that are required to be knowledgeable of the CPU burst time ahead of time [7] [6].

To help in predicting correctly, a machine learning based strategy was implemented using a subset of data from the "GWA-T-4 AuverGrid" computational grid data set [1]. Our implementation on the prediction of process burst time will be more extensive than previous work. Helmy et al. used only the first 5000 jobs to estimate the CPU burst time and no explanation was provided for not using the entire data set with 414,176 jobs. Our research project will try to answer the following research questions:
RQ-1. Does the performance increase or decrease when the machine learning algorithms include all the jobs as input data?
RQ-2. Does normalizing job features impact the accuracy of the prediction of CPU burst time?

RQ-3. Do other machine learning algorithms perform better than the implemented machine learning algorithms in [1]?

II. Related Works

As mentioned in the previous work by Helmy et al. [1] some scheduling algorithms require knowing the CPU burst times for processes sitting in the ready queue. For these scheduling algorithms that require CPU burst times beforehand, such as SJF and SRTF, A mathematical formula exists [8] which can assist in estimating the length of a process's next CPU burst. The formula is as follows, Given n process $P_1, P_2, ..., P_n$ and the bursts time of every processes as $T_1, T_2, ..., T_n$, anticipated burst time for process P_{n+1} is given as

$$\tau_{n+1} = \alpha T_n + (1 - \alpha)\tau_n \tag{1}$$

where τ is the predicted CPU Burst time. This is known as "Exponential averaging". EA is described by Vandana [9] as "*To forecast the next CPU-burst time of a process which is a kind of time series, we use average exponential formula. This formula is based on the history of past CPU-burst times of the processes which have been executed by the processor.*"

Furthermore, in the process of knowing CPU burst times and the methods to use for the same, there is another proposed method [4] which is called the Dual Simplex optimization technique. Using this method, it is possible to determine the length of the next CPU burst. In this proposed work, It is mentioned that their calculations indicate how there is similarity between the next CPU burst time and the previously executed processes' CPU burst time, allowing the process with the shortest predicted CPU burst time to be selected. This proposed method was based on Linear Programming Models Dual Simplex Algorithm (LPMDSA) [10]. In the Dual Simplex method, both kinds of constraints are considered, and the required slack/surplus variables are mixed with the original *lpp* into standard form. The process starts with the standard SJF algorithm while also finding out by calculation the waiting time and turnaround time in relation to the given burst time for each process. The burst time with the turnaround and the waiting time is then turned into a mathematical formula known as Linear programming. In addition, the dual simplex approach was applied.

Another proposed method to predict CPU burst times for the SJF scheduling algorithm was by Pourali et al. [11]. This proposed approach works with an intelligent fuzzy system that uses the past behavior of a process to predict the time taken in execution (next CPU-burst time) with more efficiency. A rule system based on knowledge was used, it is called a fuzzy system [12]. An integral part of this system is the database which is based on a lot of if-then rules. The way that this approach works is we have a system input and a system output. The input is the value of the previously used CPU-burst of a process, whereas the output is the next anticipated CPU-burst time. Major advantages of using this system is high flexibility and preferred calculation speed. In addition, prediction of future run time with the help of historical information for process execution is helpful and described by Smith et al. [13] as "*A technique for deriving predictions for the run times of parallel applications from the run times of 'similar' applications that have executed in the past. The novel aspect is the use of search techniques to determine those application characteristics that yield the best definition of similarity for the purpose of making predictions.*" The aforementioned search techqniques used in this paper are greedy search and genetic algorithm search. In conclusion, they found that genetic search always performed better than greedy search for every workload.

Machine learning algorithms have also been previously used for Operating systems concepts. Negi et al. [14] implemented a Machine learning algorithm-based CPU scheduler to reduce the Turn Around Time (TAT) of a process in the Linux Operating system. The concept of machine learning helped in finding out the CPU time-slice utilization behavior in the operating system of the previously known programs. Finding the most crucial static and dynamic attributes of the processes was their main goal. These processes help in predicting CPU burst times. These dynamic attributes were used as input such as input size, program size, read-only data size, and so on. In conclusion. they found out that the decision tree algorithm was the most effective. Moreover, in another work presented by Shulga et al. [15], a scheduler is proposed which selects the executing device(between CPU and GPU) after prior training, This decision is influenced on the size of the input data. As making optimal use of all available computing devices is critical issue to be solved for heterogeneous computing systems. The system performance can be heavily influenced by the selection between a CPU or a GPU processor. They were able to successfully classify tasks between CPU and GPU using the support vector approach after just a few runs of computing cores were learnt. This ML-based approach helped in achieving a significant reduction in the time of processing a set of tasks, and high accuracy of classification.

III. Proposed Approach

The workflow of our proposed approach is shown in Fig. 1 within blue region. The ready queue and CPU task is for the Grid Computer to handle.

First, we have shuffled the whole dataset so that machine learning classifiers do not over-fit and remain general. The shuffling also ensures that the training and testing of the data will have a normal distribution of data.

Then, if any feature value is missing for all the instances, we have removed the feature. The following features were removed because they only contained the value '-1' which indicates a missing value. These are the columns that were discarded after filtering, *'JobStructure', 'JobStructureParams', 'UsedNetwork', 'UsedLocalDiskSpace', 'UsedResources', 'ReqPlatform', 'ReqNetwork', 'RequestLocalDiskSpace', 'ReqResources', 'ProjectID'*. Table I shows the dataset statistics about users, groups, jobs, or processes before and after filtering.

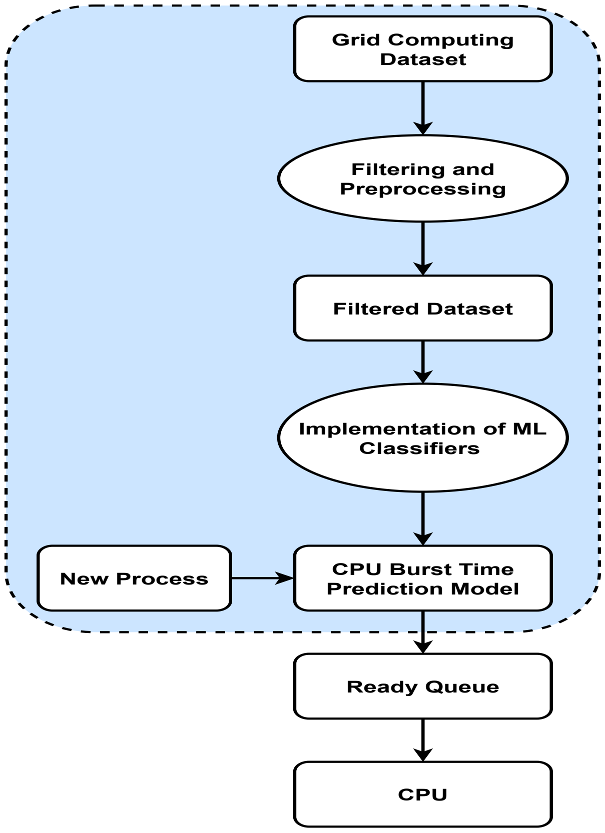

Grid Computing Dataset
Filtering and Preprocessing
Filtered Dataset
Implementation of ML Classifiers
New Process
CPU Burst Time Prediction Model
Ready Queue
CPU

Fig. 1. Proposed Approach to predict CPU Burst Time on "GWA-T-4 AuverGrid" dataset.The dotted region with blue background shows our work.

We have altered the string-based *'userID'* and *'groupID'* into vector format by using Feature Hashing as a hash function in order to use the identifiers as features. The *'userID'* is now 10-dimensional vector so that it can accommodate $2^{10} = 1024$ users and the *'groupID'* is 4-dimension with capacity of $2^4 = 16$. This is more than enough user and group space needed for our dataset. We have considered more capacity for these two features, so that in the future new users and groups can be added for training or testing.

Normalizing or scaling features on a dataset can improve the performance of certain ML algorithms like K-Nearest Neighbors, K-Means clustering [16] [17]. For normalization, we have used min-max scaling.The formula can be represented mathematically by equation (2). Here, x' is the scaled value, x is the actual value, $min(x)$ is minimum value of the column or feaute, and $max(x)$ is the maximum value of the feature.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{2}$$

After the prepossessing of data, the next phase was to train and test the dataset, for which we implemented multi-variate machine learning algorithms [18] where prior to training we randomized the dataset again to prevent the ML algorithms from biased learning order of training set. To make the data more generalized we used k-fold cross validation strategy which is a data partitioning strategy for ML algorithms [19]. For our experiment, we have set $k = 5$. In this approach, 80 percent of the data is used to train while 20 percent is used to test the model. Details of the 5-fold cross-validation is shown Fig. 2.

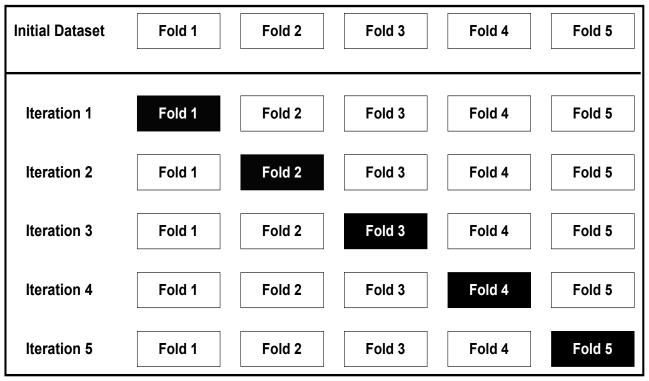


Fig. 2. 5-fold Cross-Validation Process. Initially the dataset is split into 5 similar length folds. There will be 5 iterations of ML algorithms. In each iteration one fold will be the test set and the other four folds will be the training set. Black folds indicate the test data while white folds represent training data.

TABLE I
DATASET DIFFERENCE BEFORE AND AFTER FILTERING.

	Before Filtering	**After Filtering**
Total Jobs	404,176	347,611
Total Users	406	404
Total Groups	9	8

IV. Dataset Description

AuverGrid is a production grid platform made up of five clusters spread over the central region of France in Auvergne. Clusters consists of twin 3GHz Pentium-IV Xeon nodes which is responsible for hosting Scientific Linux to perform complex computing. The Auvergrid project, which is part of the EGEE project [20], is a regional grid. The LCG middleware is used as the grid's infrastructure. Its primary applications are biomedical and high-energy physics research [21]. Table II describes the "GWA-T-4 AuverGrid" dataset features that we used in our experiment.

Group-wise number of users and numbers of jobs are shown in table III. Group 3 has the highest number of users as well as the highest number of jobs whereas group 8 has only 8 users and group 5 has only 9,702 processes recorded as the lowest for users and jobs.

V. Experiments and Results

We have used python version 3.7 for our experiment. The machine learning algorithms were implemented using Sckit-learn [22], an open-source machine learning library. All of our

TABLE II
LIST OF ATTRIBUTES IN THE DATASET

Serial	Attribute Name	Attribute Description
1	Job	Identifier of Job count
2	SubmitTime	Submission time (seconds)
3	WaitTime	Difference of SubmitTime and Time at which it started running (seconds)
4	RunTime	The ongoing time when job is running (seconds)
5	NProcs	Number of processors the ongoing Job is using
6	AverageCPUTimeUsed	Total CPU time used by all the jobs divided by number of allocated processors.
7	Used Memory	Average used memory per processor in kilobytes.
8	ReqNProcs	Amount of processors requested
9	ReqTime	Requested time measured in seconds. can be run time or avgerage CPU time per processor
10	ReqMemory	Requested memory (average per processor) per processor in kilobytes.
11	Status	Determining the status of Job completed=1, failed=0, cancelled=5.
12	UserID	String identifier for users executing Jobs
13	GroupID	String identifier for groups users belongs to
14	ExecutableID	Name of executable (application), a natural number, between one and the number of different applications appearing in the workload
15	QueueID	Identifier for queue, String format
16	PartitionID	To identify which machines in the cluster were used (String)
17	OrigSiteID	Describes the origin form where the Job was originated, to distinguish between them. (String)
18	LastRunSiteID	The most recent site of the Job run (String)

experiments were done on Google Collaborator [23], an online python code compiler developed by Google.

We used seven different ML algorithms to predict CPU Burst length. For Linear Regression (LR), Gradient Boosting (GB), and Extreme Gradient Boosting (XGB) we have used the default parameters provided by Scikit-learn. Our Support Vector Machine (SVM) was tuned with value C and $Maximum\_Iteration$. The C value for SVM is a regularization parameter, that tells the SVM optimizer how much we want to avoid misclassification in our training set [24]. To ensure optimized time for SVM we have set the $Maximum\_Iteration$ to 10,000 so that SVM can perform the regression in a reasonable duration. For K-Nearest Neighbors (K-NN) model, the parameter $N\_Neighbors$ was set to 5, means 5 nearest neighbors were considered for the output of a test example. Both Random Forest (RF) and Decision Tree (DT) algorithms were tuned only by $Maximum\_Depth$ of the tree.

TABLE III
GROUP-WISE USERS AND JOBS OF AUVERGRID DATASET

Group No.	Number of Users	Number of Jobs
G1	109	24,311
G2	46	37,792
G3	159	145,508
G4	47	88,681
G5	10	9,702
G6	16	15,924
G7	11	13,790
G8	8	11,903

We have evaluated our ML models based on an average 5-fold cross-validated Coefficient of Determination (CD) also known as the R2 Score. The best possible score for CD can be 1 and the worst can be even negative if the model performs very poorly. We can represent CD mathematically by the following formula (3) where, y indicates the actual real value, $\hat{y}$ represents the predicted value by ML algorithm, and $\bar{y}$ is the average of all real values. The result of the machine learning models are shown in Table IV.

$$CD(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3}$$

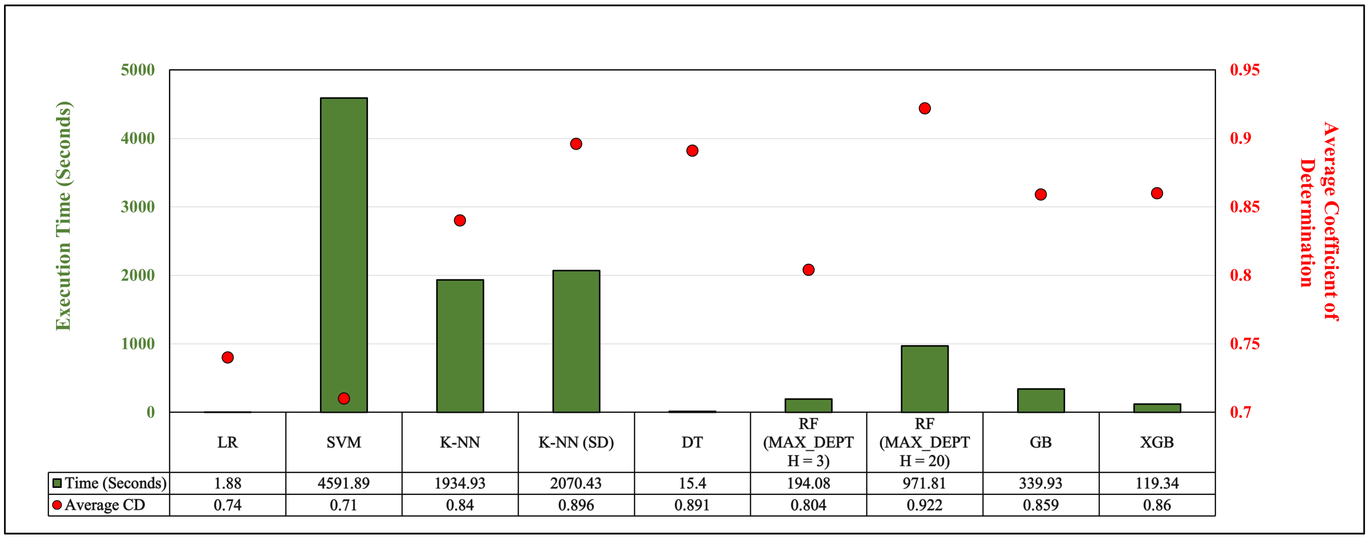
TABLE IV
COEFFICIENT OF DETERMINATION COMPARISON AMONG DIFFERENT MACHINE LEARNING ALGORITHMS ON GWA-T-4 AUVERGRID DATASET

Estimator	Parameters	Average Coefficient of Determination (CD)
Linear Regression	Default	0.740
SVM	C=100, Maximum Iteration = 10,000	0.710
K-NN	N_Neighbors = 5	0.840
		0.896^{SD}
Decision Tree	Max_Depth = 20	0.891
Random Forest	Max_Depth = 3	0.804
	Max_Depth = 20	0.922
Gradient Boosting	Default	0.859
XGBoost	Default	0.860

^{SD} Using the Scaled Dataset.

From Table IV we can observe that Random Forest with Max_Depth of 20 achieves the best average CD of 0.922 while Support Vector Machine performs worst with the CD value of average at 0.71. The low score by SVM is expected due to its nature of poor performance on large datasets [25].

Fig. 3, presents a combined Bar and Scatter plot where the execution time of each estimator for 5-fold cross-validation is represented by a bar graph and their average coefficient



	LR	SVM	K-NN	K-NN (SD)	DT	RF (MAX_DEPTH = 3)	RF (MAX_DEPTH = 20)	GB	XGB
🟩 Time (Seconds)	1.88	4591.89	1934.93	2070.43	15.4	194.08	971.81	339.93	119.34
● Average CD	0.74	0.71	0.84	0.896	0.891	0.804	0.922	0.859	0.86

Fig. 3. Execution Time and Average Coefficient of Determination (CD) score comparison of different ML Algorithms on GWA-T-4 AuverGrid Dataset. Left side of Y-axis indicates Time in seconds. Bar graph in light green color indicates the time for each classifier. Right side of Y-axis indicates average coefficient of determination (R2 Score). The red dots represent the average CD for individual classifier.

correlation value is shown in the scatter plot. The left side of Y axis indicates the Time in seconds and the right side of Y axis denotes Average CD. If time constrain are a factor for Grid Computing, we can easily recommend a Decision Tree classifier as it takes only 15.4 seconds for 5-fold cross-validation with an average coefficient correlation of 0.891. In the case of the most accurate estimation, Random Forest with Max_Depth of 20 is recommended. Another interesting finding is that Linear Regression model only takes 1.88 seconds. LR is the fastest classifier among all the algorithms that even outperforms Support Vector Machine.

One of the disadvantages of our present approach is how will the estimator predict the CPU Burst time for a new user or a new group. One solution can be given as if a new user comes to an existing group, prediction for the first time will be based on the average or median CPU Burst time of that group. If a new user comes with a new group then the average or median CPU Burst time of the entire dataset can be used for predicting. Further investigation on this topic is needed for better prediction is a task for future work.

Coming back to our research question, the first RQ was to investigate the output while including all the jobs from the dataset. Answer to this question is in Table V. No, including all the jobs will not improve the correlation coefficient. But, this approach result is more robust and more accurate for real-life cases. Only taking 5,000 jobs by Helmy et al. [1], represents a partial scenario of the dataset. Also, their approach did not mention any reason behind taking the first 5,000 jobs which consists of only 1.2% of the entire dataset. Even though our proposed approach did not outperform Helmy et al., our proposed approach is more sustainable and this approach can precisely predict CPU Burst time.

Our second research question was about the impact of normalization on features. Normalizing the features, in fact, performs better. K-Nearest Neighbors classifier with scaled data achieves 6.67% better result than without scaling. The overhead time for scaling data was only 1.3% more than non-normalized data. Table VI shows the the comparison between normalized and non-normalized dataset for K-NN algorithm. All the other machine learning models performed same with and without data normalization. This is because of the fact, not all the ML estimators can not take advantage of normalization.

TABLE V
COMPARISON OF AVG. CD BETWEEN OUR APPROACH AND HELMY ET AL. [1]

Estimator	**Our Approach**	**Helmy et al. [1]**
K-NN	0.896	0.956
SVM	0.710	0.953
Decision Tree	0.891	0.943

TABLE VI
COMPARISON BETWEEN NORMALIZED AND NON-NORMALIZED DATASET ON K-NN

Estimator	**Normalization of Data**	**Average CD.**	**Execution Time (Seconds)**
K-NN	FALSE	0.840	1934.93
K-NN	TRUE	0.896	2070.43

The final research question is about the performance of different machine learning methods for the task of predicting CPU Burst time on the Grid system. The four machine learning algorithms used by Helmy et al. [1] are SVM, K-NN, Random Forest, and Multi-Layer Perceptron. We have deployed four other ML algorithms like Linear Regression, Decision Tree, Gradient Boosting, and XGBoost. Except for

Linear Regression, the performance of all the other classifiers are comparable in terms of Correlation Coefficient and Time. One of the significant analysis results is that Decision Tree Algorithm takes very little time to complete the task while performing average CD of 0.891. In a time constrained Grid system, DT will generate highly accurate CPU burst time faster than most of the ML algorithms.

VI. Future Work

Investing more in the ML classifier's parameters may increase the performance. As the Random Forest classifier has a 14.67% increase in coefficient of determination when the maximum depth is set to 20 instead of the default 3. In our future work, we will dive deeper into these parameters to analyze the performance of these classifiers. We can get benefit using a Grid Search that can search exhaustively over predetermined values for estimators.

Moreover, Artificial Neural Networks (ANN) are getting more popular now than ever. Neural Networks perform better for large dataset and can achieve state of the art results. We will incorporate the ANN to predict the grid CPU burst time in our future work.

Another interest for the future lies in user based CPU burst time prediction. Instead of looking into the whole grid dataset, we want to analyze the time-series data for CPU burst for individual user in the grid and estimate the burst time based on their past usage behavior.

VII. Conclusion

We suggested a strategy to predict the CPU burst length for processes in the ready queue for Grid Computing utilizing Machine Learning techniques such as LR, SVM, K-NN, DT, RF, GB, and XGB in this research. The characteristics feature selection approaches used in the proposed methodology were implemented for grid workload designated "GWA-T-4 AuverGrid". The experimental findings demonstrate a strong, linear link between process characteristics and burst CPU time, with Random Forest (Max_Depth = 20) outperforming other ML approaches in terms of CD. Our recommendation for the optimal performance is the Decision Tree algorithm, which is 63x times faster than best performer Random Forest while the R2 score is only 3.25% decreased. Also, in the case of fastest output, Linear Regression can achieve the fastest solution but the output accuracy may not be the best.

Furthermore, ML algorithms can take advantage of feature normalization or scaling for prediction. For instance, K-NN classifier performed better with feature scaling. Even though all the other ML classifiers did not have any effect on feature scaling, we would still recommend normalizing the dataset so that the algorithms which can take advantage of this approach, can perform better.

Acknowledgment

The authors would like to thank the AuverGrid team as well as the owners of the AuverGrid system, for publishing the "GWA-T-4 AuverGrid" dataset.

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," in *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, 2015, pp. 3–8.

[2] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts*, 8th ed. John Wiley & Sons, 2013.

[3] H. Amur, G. Shenoy, D. Sarma, and S. Vaddagiri, "Plimsoll: a dvs algorithm hierarchy," 01 2008.

[4] M. R. Mahesh Kumar, B. R. Rajendra, C. K. Niranjan, and M. Sreenatha, "Prediction of length of the next cpu burst in sjf scheduling algorithm using dual simplex method," in *Second International Conference on Current Trends In Engineering and Technology - ICCTET 2014*, 2014, pp. 248–252.

[5] Microsoft, "What is grid computing?" Accessed March 25, 2022. [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-grid-computing/

[6] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[7] S. Shah, A. K. Mahmood, and A. Oxley, "Analysis and evaluation of grid scheduling algorithms using real workload traces," 01 2010, pp. 234–239.

[8] A. Singhal, "Predicting burst time : Sjf scheduling," 2022. [Online]. Available: https://www.gatevidyalay.com/predicting-burst-time-sjf-scheduling/

[9] Vandana, "Revised formula for estimating cpu burst," *Journal of International Academy of Physical Sciences*, vol. 22, no. 4, pp. 345–354, 2018.

[10] H. S. Kasana and K. Kumar, "Introductory operations research," 2004.

[11] A. Pourali and A. M. Rahmani, "A fuzzy-based scheduling algorithm for prediction of next cpu-burst time to implement shortest process next," in *2009 International Association of Computer Science and Information Technology - Spring Conference*, 2009, pp. 217–220.

[12] L. A. Zadeh, "Fuzzy sets," in *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*. World Scientific, 1996, pp. 394–432.

[13] W. Smith, I. Foster, and V. Taylor, *Predicting application run times using historical information*, 07 2006, vol. 64, pp. 122–142.

[14] A. Negi and P. K. Kumar, "Applying machine learning techniques to improve linux process scheduling," in *TENCON 2005 - 2005 IEEE Region 10 Conference*, 2005, pp. 1–6.

[15] D. A. Shulga, A. A. Kapustin, A. A. Kozlov, A. A. Kozyrev, and M. M. Rovnyagin, "The scheduling based on machine learning for heterogeneous cpu/gpu systems," in *2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, 2016, pp. 345–348.

[16] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494619302947

[17] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *CoRR*, vol. abs/1503.06462, 2015. [Online]. Available: http://arxiv.org/abs/1503.06462

[18] K. Fauvel, V. Masson, and É. Fromont, "A performance-explainability framework to benchmark machine learning methods: Application to multivariate time series classifiers," *CoRR*, vol. abs/2005.14501, 2020. [Online]. Available: https://arxiv.org/abs/2005.14501

[19] D. Anguita, L. Ghelardoni, A. Ghio, L. Oneto, and S. Ridella, "The 'k'in k-fold cross validation," in *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. i6doc. com publ, 2012, pp. 441–446.

[20] "Enabling grids for e-science in europe." [Online]. Available: http://public.eu-egee.org/intro

[21] "The grid workloads archive." [Online]. Available: http://gwa.ewi.tudelft.nl/datasets/gwa-t-4-auvergrid

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] E. Bisong, *Google Colaboratory*. Berkeley, CA: Apress, 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-4470-87

[24] M. S. (https://stats.stackexchange.com/users/9196/marc shivers), "What is the influence of c in svms with linear kernel?" Cross Validated, uRL:https://stats.stackexchange.com/q/31067 (version: 2012-08-23). [Online]. Available: https://stats.stackexchange.com/q/31067

[25] W. H. Khoong, "When do support vector machines fail?" Aug 2021. [Online]. Available: https://towardsdatascience.com/when-do-support-vector-machines-fail-3f23295ebef2