EXTES: An Execution-Time Estimation Scheme for Efficient Computational Science and Engineering Simulation via Machine Learning

Seounghyeon Kim\(^1\), Young-Kyoon Suh\(^1\), and Jeeyoung Kim\(^2\)

\(^1\)School of Computer Science and Engineering, Kyungpook National University, Daegu, South Korea
\(^2\)Center of Self-Organizing Software Platform, Kyungpook National University, Daegu, South Korea

Corresponding authors: Young-Kyoon Suh (e-mail: yksuh@knu.ac.kr) and Jeeyoung Kim (e-mail: jeeyoungkim@knu.ac.kr).

This work was supported by the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (21A20131600005), the National Research Foundation of Korea(NRF-2018R1C1B6006409), Basic Science Research Program through the NRF funded by the Ministry of Education(NRF-2018R1A6A1A03025109), and the Institute of Information Technology Planning(IITP) grant funded by the Ministry of Science and ICT(No, 2016-0-00145).

ABSTRACT In recent years, computational science and engineering (CSE) simulations using high-performance computing resources are actively exploited to solve complex domain-specific problems. Thanks to the remarkable advance of IT technology, the CSE community is challenging more complex and difficult problems than ever before, by running these simulations online. In this regard we often witness that (i) online simulation users suffer from knowing little about the estimated termination time of their launched simulations, and (ii) the limited computing resources are squandered by wrong input that leads the simulations to run forever. To address such issues, we propose a novel execution time estimation scheme, termed EXTES, using machine learning techniques for more efficient online CSE simulations. With a large amount of existing provenance data, the EXTES scheme trains a suite of models rooted from classification, regression, and a hybrid of the two and utilize these models to estimate the execution time for specified input parameters for simulations. In the experiments on real simulation data, our proposed models achieved about 73% accuracy on average in execution time estimation across sixteen simulation programs taken from a variety of CSE fields. In the meantime, the overhead incurred by the training and estimation is almost negligible.

INDEX TERMS Computational Science and Engineering Simulation, EDISON, Execution Time Estimation, Machine Learning, Provenance Data

I. INTRODUCTION
Computational science and engineering (CSE) is a field where high-performance computing resources are used to solve complex problems, unlike other fields conducting research by performing physical experiment or meticulous theoretical analyses. With the consistent advancement of IT technology, the CSE field is now able to solve more complex and difficult problems than ever before.

One example of said high-performance computing resource is EDISON [1,2]—an online platform that shares various CSE simulation programs. EDISON allows users to easily execute simulations. More often than not, many simultaneous simulations performed on the EDISON platform require a huge amount of execution cost, taking exceedingly long times—days, weeks, and even months—depending on the input parameters specified by the simulation. Erroneous parameters can cause problems resulting in increased cost of executions to no avail, which leads to wasting away the limited resources that are shared among a number of simultaneous users—resources that could have been put to better use.

Also, those who are unfamiliar with EDISON may not be able to adequately estimate the amount of time their simulations may take. These novice users may wait long periods of time not knowing when their simulations will return results, which makes it difficult for them to design valid research plans. It is not uncommon for CSE researchers to suffer from being behind their research
schedule, due to long delays and large costs of execution. Therefore, there has been a growing need to transform EDISON to become a more intelligent and smarter platform, by satisfying both (i) platform administrators who want to manage the limited computing resources more efficiently and (ii) most users who have grievances in waiting long periods of time for their simulations.

To address these concerns, in this article we propose a novel method using machine learning techniques [3,4] to estimate the completion time of a simulation in advance. More specifically, a suite of machine learning methods—support vector machines (SVMs), random forest, decision tree, and k-nearest neighbors (k-NN)—are trained on a total of 61,000 provenance records that we gathered from a variety of (sixteen) simulation programs conducted on EDISON. These machine learning models are then utilized to predict the execution time of a simulation with its specific input parameters. Using our proposed models, EDISON users and administrators alike could detect any potential risks (e.g., infinite loops or unexpected quick termination) ahead of time. This will benefit the users and administrators by allowing them to design best simulation schedules and conserve limited resources by not running any simulations with known risks.

The academic contributions of this article are summarized as follows:

- We propose an ensemble method to estimate simulation run-time in advance, by using statistical machine learning techniques applied to real provenance data acquired from a variety of sixteen HPC simulation programs in service on EDISON.
- We show that even with a handful of simulation parameters, the running time of a simulation program can be predicted with high accuracy.
- We evaluate the performance of our proposed algorithms on actual simulation data performed on the EDISON platform.
- We further conduct statistical analyses on potential factors affecting simulation execution time estimation based on input parameters.
- We suggest future research directions.

This manuscript is a substantial extension of an earlier work [5]. The organization of this manuscript is in the following. The following section discusses related work. In Section III, we conduct an in-depth analysis on the simulation provenance data used throughout this manuscript. Then, in Section IV we give a description of our approach for predicting execution time. In turn, Section V presents our evaluation results. Finally, Section VI concludes this manuscript and discusses future work.

II. RELATED WORK

The EDISON platform [1, 2] is a well-known system, enabling online simulation, as mentioned earlier. EDISON is actively used by dozens of schools and numerous users in Korea and some other countries like Taiwan, USA, etc.

Lee et al. [6] introduced a system that uses machine learning techniques to infer the “results of a simulation” for a given input parameter(s). Their work is fundamentally different from ours, in that the previous work foresees the outcome of a simulation based on the output data generated from the chosen simulation program while our work predicts the run time of the simulation, although both utilize the information of input parameters of the program for estimation. Moreover, the study is limited to only one simulation program, but our study is very comprehensive in that it covers a variety of simulation programs.

The authors [7] conduct a study that predicts through the Support Vector Regression (SVR) technique based on the work-time clustering of parallel programs. Though their work is similar to our work in light of predicting running times via machine learning models, the interests of the authors are limited to only applying clustering techniques, and the scope of their work lies in the running time of “parallel programs” unlike ours.

Another study [8] also uses machine learning techniques to predict the running time of GPU applications. Some common models like support vector machines and random forests are used in their work, but the application domains of that study are GPU-based applications, which are different from ours.

On one hand, a similar former study [9] on EDISON was conducted to attempt to predict simulation execution time based on given input parameters as well. As opposed to our work, however, that study is limited to the programs only from the field of computational fluid dynamics (CFD). On the other hand, our proposed models are general-purpose: that is, the proposed models are applicable to any arbitrary program (taking input parameters) not only from CFD but also from other CSE areas such as Nano Physics (NANO), Computational Chemistry (CHEM), Computational Medicine (CMED) etc., for EDISON. Moreover, while the past study considers only regression model, our work is far more comprehensive and again general, by taking into account not only such regression model but also classification and hybrid models combining the two.

The authors [10] propose a run-time prediction method to improve the performance of scheduling algorithms. As with the aforementioned study [9], this past study considered only regression model. In addition, the authors’ research predicts execution times based on application similarity but not on a huge amount of provenance data collected from simulation programs as does our work. Furthermore, while the previous study [10] aims to better the performance of scheduling algorithms, the ultimate goal of our study is to reach the enhanced efficiency of CSE simulations by foreseeing their execution times.

In another study [11], the authors aim to support the execution of workflows in computational science. They propose a method to profile workflows such as runtime,
This section analyzes the provenance data collected from EDISON and briefly describes our approach for estimating simulation execution time based on the analysis.

A. ANALYSIS OF SIMULATION PROVENANCE DATA

The EDISON platform offers a variety of simulation programs (hereinafter simulation apps) developed in various fields of CSE. About one-year (from August 2016 to July 2017) worth of simulation provenance data were used for our study and gathered from a set of some representative simulation apps actively used in CFD, CHEM, CMED, and NANO fields in conjunction with EDISON.

Table I provides the summary of provenance data from the aforementioned sixteen apps. In actuality, an individual provenance record mainly consists of the name of a specific area of CSE, the name of a simulation app, and the execution time spent for that simulation. Indeed, the record also includes the parameter value(s) entered by the user when that simulation was executed although the specific values are not shown in the table.

Table I also shows a variety of statistical information, including the simulation counts, the size of provenance data in Kilobytes (KB) as well as the minimum, maximum, and average run times in seconds, for each app. The simulation run time was typically over 2 seconds, and the longest run time for each simulation was different; in some cases it was more than 2.3 million seconds (about 26 days or about a month) for the longest run time. In some other cases, the simulation run times were under 2 seconds; actually, these cases were excluded from our valid dataset as they were considered as simulation failure caused by an execution error. The overall average run time across these apps was about 3 hours.

Here are a couple of reasons why the simulation apps are selected. First, they are commonly used by EDISON users as shown in the table. One of these reasons is that both 2D_Incomp_P and WaveSimulation showed that most run times were concentrated within a very short period of time in the entire time range. In addition, the provenance records, which required a very long run time, were extremely rare. Considering these time distributions, we shaped our idea of estimating a run time by interval.

Table I provides the summary of provenance data obtained from a set of some selected apps. The table also shows different fields that work with estimation and big data such as image processing and classifications for extreme environments [12,13] and various AI domains as well [14,15].

III. OUR APPROACH

This section analyzes the provenance data collected from EDISON and briefly describes our approach for estimating simulation execution time based on the analysis.

memory usage, and CPU utilization and to automatically characterize workflow work requirements based on the profiled data. Their study monitors the execution of the workflow and estimates the amount of memory consumption in addition to the execution time estimate. Unlike their work [11], our simulation runtime estimation method is more lightweight as there is no requirement for such a heavy profiling process or simulation monitoring.
On one hand, the information we could use to estimate the execution time was limited to only user input parameter information. Such constraints led us to consider various machine learning models that could be trained to predict run time using input (i.e., parameter information) and output (i.e., execution time) data.

Based on these analyses and observations, the following subsection describes our execution time estimation approach via machine learning techniques considered in this article.

**B. TIME-DISTRIBUTION BASED SIMULATION RUN TIME ESTIMATION USING MACHINE LEARNING TECHNIQUES**

As discussed in Section III.A, given that execution times are concentrated in a short period of time rather than evenly distributed, we consider employing classification approach. More specifically, for a simulation app we divide the entire range of its (past) execution times into 100 subranges (or intervals), and the expected simulation time for its specified simulation input parameters is reported in intervals. In other words, this method is seen as a classification technique selecting a labeled interval.

For the classification, we consider two classic machine learning models [4]. One is decision tree, and another is k-nearest neighbor (k-NN) techniques. Our proposed algorithm selects the classification model with better accuracy between the two techniques.

However, an ironical disadvantage of a classification model is that the model reports the estimated execution time as an interval. It is evident that the interval-based time reporting gives approximation to users, but it is not practically precise.

To compensate for such a problem, we also consider employing regression approach as an alternative, thereby making more accurate time estimates feasible. For the regression, we utilize decision tree, random forest, and support vector machine (SVM) techniques [4]. Our proposed algorithm selects the regression model with the best accuracy among the three techniques.

Not surprisingly, regression-based time estimates could also expose critical drawbacks. Estimated times by regression are typically more precise but less accurate than those by classification. In other words, it could be very difficult for the regression method to predict the “exact” time of simulation execution for given input parameters.

All things considered, the classification can predict the execution time interval with better accuracy while the regression the time as a point more precisely.

Now the following conclusions can be drawn about the classification and regression. If we divide the entire time interval into small time intervals, apply the classification to the divided intervals, and then perform the regression within the guessed interval (by the classification), then the error rate of the regression can be dropped on average. Therefore, we propose a “hybrid” model that inherits the merits of both of the classification and regression.

Our hybrid model divides the entire range of simulation run times into 10 subranges for a particular simulation app and performs the regression within the interval selected by the classification for given simulation input parameters. Namely, we estimate a certain time interval based on the specified input parameter values (by the classification) and then calculate the expected run time by the regression within that (estimated) interval. The advantage of this “ensemble” method is that the estimated times can be more accurate by considering fewer intervals than pure classification.

Considering the two facts that (i) as shown in Table I, the minimum values of the execution times are not significantly different across the simulation apps and (ii) are negligible compared to its maximum values, the errors of the regression and hybrid models can be calculated by the following formulas:

$$\text{The maximum error of regression model} = MT \quad (1)$$
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2929800, IEEE Access

The maximum error of hybrid model

\[
\text{The maximum error of hybrid model} = \left( k \frac{MT}{10} + \frac{MT}{10} \right) - k \frac{MT}{10} = \frac{MT}{10} \quad (2)
\]

\(MT\): the maximum execution time, \(k\): a classification interval \((1 \leq k \leq 10)\)

Based on the above equations (1) and (2), we can see that the error of the two models can be reduced by 10 times compared to \(MT\)—the maximum error of the existing regression model.

The hybrid model finally outputs a point of time as does the regular regression. The classification with better accuracy and the regression with less accuracy but more precision can work in concert as complementary time estimation techniques. Indeed, the hybrid model is treated as a sort of “regression” model because again its final answer is reported as a point of time, and thus, it is proper to compare its performance with that of the normal regression.

Last but not least, when estimating the time to run the simulation, we provide both results: (i) the estimated time by the classification model and (ii) the estimated time by the regression or hybrid model (whichever provides a better result). By doing so, the EDISON users can decide whether or not to continue with their simulations.

**IV. SIMULATION RUN TIME ESTIMATION ALGORITHM**

This section describes the actual implementation and details of the algorithm proposed in Section III.B.

A. TRAINING EXECUTION TIME ESTIMATION MODELS

Table II presents an algorithm for training the models used to estimate execution time for a simulation app. In the following subsections, we elaborate on each major part of this algorithm.

1) DATA REFINEMENT

In Table II, lines 1-3 correspond to the process of refining simulation execution provenance data of the EDISON platform. The input data (\(\text{input\_data}\)) is structured as described in Section III.A. The proposed algorithm uses \(\text{input\_data}\) that includes user input parameter information and corresponding execution times. The input parameter information consists of from at least two to dozens of numeric or categorical attributes. Execution time is an attribute with an integer value in seconds. Since there are many kinds of simulation apps available on EDISON, it is almost infeasible to apply a suitable data refinement algorithm to each simulation app. Instead, we perform a universal data refinement process for all the sixteen simulation apps. In this article three methods are applied.

| Input: \(\text{input\_data} \leftarrow \) Simulation parameter values and runtimes data |
| Output: \(\text{classification model, regression model}\) |
| \(\text{classification\_model\_list} = \{\text{ctree, ksvm, knn}\}\) |
| \(\text{regression\_model\_list} = \{\text{randomForest, ctree, ksvm}\}\) |
| Line 1: \(\text{Refine} \text{\_\_data}\) |
| Line 2: \(\text{training\_data} = \text{input\_data}[0:80]\) |
| Line 3: \(\text{testing\_data} = \text{input\_data}[81:100]\) |
| Line 4: \(\text{for each model} \ m \in \{\text{classification\_model\_list, regression\_model\_list}\}\) |
| Line 5: \(\text{t} = \text{training}(m, \text{training\_data})\) |
| Line 6: \(\text{accuracy} = \text{validation}(t, \text{testing\_data})\) |
| Line 7: \(\text{if} (\ m \in \text{classification\_model\_list})\) |
| Line 8: \(\text{Output\_cl} \leftarrow \) a model with the lowest error rate |
| Line 9: \(\text{if} (\ m \in \text{regression\_model\_list})\) |
| Line 10: \(\text{Output\_rg} \leftarrow \) a model with the lowest error rate |
| Line 11: \(\text{end for}\) |
| Line 12: \(\text{Classify} \text{\_\_data} \text{into 10 time intervals}\) |
| Line 13: \(\text{Classify} \text{\_\_data} \text{into 10 time intervals through}\) |
| \(\text{Output\_cl}\) |
| Line 14: \(\text{for each time interval} \ i \ in \ (1:10)\) |
| Line 15: \(\text{result} = \text{result} + \) the relative error in regression with \(\text{(training\_data and testing\_data in } i)\) as done in lines 4 to 11 |
| Line 16: \(\text{end for}\) |
| Line 17: \(\text{result} = \text{result} / \) the number of time intervals |
| Line 18: \(\text{Output\_rg} = \text{min}(\text{Output\_rg, result})\) |
| Line 19: \(\text{return} (\text{Output\_cl, Output\_rg})\) |

The first method is outlier removal. There is some abnormal provenance data collected from \(i\) simulations terminating in a very short time due to an internal problem caused by the EDISON platform or by simulation apps themselves and \(ii\) simulations taking forever. We therefore regard such erroneous data as outliers and remove it.
Specifically, we obtained the average value of the execution times per app and discard the provenance data with the one that is farthest away from the average. However, if we find an elbow at which a large amount of the data is discarded by the outliers, or the number of the remaining records drops below a threshold, then we stop removing the outliers.

The second method of refining is to remove the execution times associated with duplicate parameter values. Even though the parameters are the same, and the programs are the same, the execution times may vary by external factors (i.e., hardware, simultaneous simulations, etc.). The variation could be a very fatal noise for run time prediction. To deal with this noise, we compute the average of the run times with the same input parameters for a simulation app and substitute all the run times by the computed average.

The last method of refinement is normalization. Because each parameter has a wide numerical range, the difference in the range can cause noise. To get rid of this noise, the execution times were normalized between 0 and 1. The applied normalization formula is as follows:

$$x_i = \frac{x_i - \min(x)}{\max(x) - \min(x)} \quad (x: \text{an execution time value}) \quad (3)$$

Nevertheless, the data refinement techniques we applied do not guarantee high accuracy for all the simulation apps. (It seems that further research is still needed to investigate why.) In addition, we also have found that the order of the three refining processes can also affect accuracy. We thus devised a total of nine combinations of the three refinement methods, and then we used the combination method with the best performance.

2) CLASSIC MODEL TRAINING AND ACCURACY MEASUREMENT

In Table II, lines 4-11 apply classic regression and classification models to conduct learning and accuracy measurements of simulation run-time forecasts. As mentioned in Section III.B, for the classification we use the decision tree (ctree) and k-Nearest Neighbor (knn) techniques, and for the regression, random forest (randomForest), decision tree (ctree), and support vector machine (ksvm) techniques.

The method for measuring the accuracy of the classification is as follows. In the classification, we divide the whole range of execution times into equal-sized 100 intervals, each of which is labeled as an index of 0 to 99. The classification model then outputs a predicted index value given a set of given input parameter values. If the predicted index value is equal to the true value (or, the execution time in seconds) corresponding to the given input simulation parameters, we consider the model to predict correctly.

Measuring the accuracy of a regression model is as follows. In the regression, for a given set of input parameter values we measure the relative error between the raw execution time and the predicted time as shown below in the following equation:

$$r = \frac{\max(x_{raw}, x_{est}) - \min(x_{raw}, x_{est})}{\max(x_{raw}, x_{est})} \quad (4)$$

$$r: \text{a relative error}, x_{raw}: \text{raw execution time} \quad x_{est}: \text{estimated time}$$

We then average all the relative errors for the model.

3) HYBRID MODEL TRAINING AND ACCURACY MEASUREMENT

Lines 12-18 in Table II show training the hybrid model, a key idea in this article, and measuring its accuracy. In line 12, the data for model training is divided into 10 intervals over the time span. In line 13, the data for model validation and testing is divided into 10 sections. Subsequently, the regression is performed in lines 14-16 as done in lines 4-11. In line 17 the algorithm collects the results from the 10 intervals and calculates the final relative error, on which it then selects the best regression model, compared with the existing regression method in line 18. Finally, the algorithm exits by yielding the best classification and regression models in line 19.

D. EXECUTION TIME ESTIMATION

Table III shows how to estimate the execution time based on the trained and validated models. When a user selects a simulation app and enters into a given set of parameter values into a specific simulation app, our execution time estimation algorithm begins, provided with the user’s parameter information, classification, regression, or hybrid model and measured accuracy. The algorithm in turn estimates the execution time on the simulation input parameter data. The user then can see the estimated time results and their accuracies by our proposed classification and regression (or hybrid) models.

V. PERFORMANCE EVALUATION

In this section we evaluate the performance of the algorithms proposed in the previous section. For our evaluation we applied a total of nine data refinement methods to the EDISON simulation provenance data and found that only “normalization” performed best on average. Therefore, we present the estimation results by the normalization.

A. EXPERIMENTAL ENVIRONMENT

The experiment took place on a desktop running the Windows 10 operating system with Intel i7-8700 CPU and 16GB RAM. Our code was written in R script [16] and executed through RStudio [17].

The following describes the packages of R that we used to apply the proposed classification, regression, and hybrid models. First, we used as the decision tree for the classification the ctree function in package party [18] among the relevant packages available in R. (The ctree function has the advantage that it does not have to go through
TABLE III
AN ALGORITHM TO PREDICT EXECUTION TIME ON A SIMULATION APP

| Input: | simulation parameter values (set by a user), classification model, regression or hybrid model, error rate of classification, relative error of regression or hybrid model |
| Output: | Predicted run time by classification, Predicted run time by regression or hybrid |

| Line1: | runtime_cl <- estimate a run time by classification model |
| Line2: | runtime_rg <- estimate a run time by regression or hybrid model |
| Line3: | return (runtime_cl, runtime_rg, error rate of classification, relative error of regression or hybrid model) |

the pruning separately because it uses the Unbiased Recursive Partitioning [19] method.) For the k-Nearest Neighbors, the knn function was used in the class package [20]. In the regression, we used the randomForest function in the randomForest package [21] for random forest, the ksvm function in the kernlab package [22] for the support vector machine, the same knn function in the class package for the k-Nearest Neighbors, and lastly, the ctree function of the party package.

B. RESULT ANALYSIS
In this section we discuss the evaluation results of our proposed models on the real simulation provenance data presented in Table I. Our analyses are conducted in various perspectives based on the experimental results that are similar to the average of the results of multiple experiments.

The data set itself used for each app was the same, but the training data and the testing data used in the experiments were randomly extracted at a ratio of 8 to 2 by changing the order of the data labels for each execution.

1) ANALYSIS OF ERROR RATES
Figs. 3 and 4 show the error rates across a few selected simulation apps in the CFD, CHEM, CMED and NANO areas.

The legends used in the figures are as follows. ctree_cl represents the results by ctree(), knn_cl by knn(), rf_rg by randomForest(), ctree_rg by ctree(), and svm_rg by ksvm(). Finally, clrg is the result of the hybrid model—the most important algorithm in our work. The y-axis in Figs. 3 and 4 represents the error rate associated with each method. In the case of classification, the results indicate the error rates, which mean that if the value is low, the interval is predicted correctly. In the case of regression, it means the average of the relative error rate. The lower the value is, the smaller the error between the raw execution time and the estimated execution time is.

The rightmost bars separated by dashes in the figures correspond to the averaged error rates among the simulation apps. Our results show that the k-Nearest Neighbors performed the best for the classification, and the hybrid model the best for the regression. We found that the hybrid method showed significant performance improvement, especially in simulations conducted on some apps like acuteSTMtip, WaveSimulation, and Single_Cell_Electrophysiology. In addition, the random forest with the best performance among the existing regression models showed a relative error rate of 29.30% on average, while the hybrid model showed better performance with a relative error rate of 27.14%.
In the case of classification, WaveSimulation showed the best performance, and the minimum error rate was as extremely low as 0.74%. Regression and hybrid models showed the best performance in dmd_pol. The average error rates were 5.15% and 5.57% in the regression and hybrid models, respectively.

2) ANALYSIS OF ELAPSED TIME FOR MODEL TRAINING AND EXECUTION TIME ESTIMATION

Table IV shows the results of elapsed times needed for model training and estimating execution time. The training time is on average 40.9 seconds per simulation.

Note, however, that this is a “one-time” cost that is not included in the actual estimation cost. Even if we should train all the suitable models for hundreds of simulation apps available on EDISON, it could take for the training to be completed within a few hours. Therefore, the overhead of model training is negligible.

The cost of estimating the execution time for the given simulation input data is 0.01 seconds on average. Considering that the minimum execution time of the simulation is 2 seconds, the overhead is extremely large, the time it takes to estimate the run time seems almost negligible.

In short, the overhead incurred by the proposed method seems almost negligible even when the simulation time estimation is in collaboration with the actual EDISON service.

3) ANALYSIS OF RUN TIME ESTIMATION CAUSAL FACTOR

We examine what factors can influence the run time estimation results for the simulation apps. The analysis focuses on the influence of classification error rates on some potential factors and on the influence of the factors on the proposed models.

For each simulation app, we explore the following potential causal factors: (i) the standard deviation of regression model, (ii) the error rate of the classification model, (iii) the average relative error of the regression models, and (iv) the degree of relative performance improvement (DPI) of the hybrid model compared with the regression model (for each simulation app). Note that the standard deviation values are normalized to a value between 0 and 1. The DPI indicates how much degree the hybrid model is superior to the best regression model with the lowest relative error on average. (Recall that we have three regression models trained by ctree, randomForest, and svm.) The DPI values can be expressed by the following equation:

\[
\text{DPI} = \frac{\text{Average Relative Error of the Hybrid Model}}{\text{min(Average Relative Errors of the Regression Model)}}
\]

Based on these potential factors, we draw two hypotheses as follows.

**Hypothesis 1 (H1):** The standard deviation of the execution time and the error rate of the classification model will have a positive correlation.

This hypothesis is driven by our observation that most simulation execution times are concentrated in a short period of time as discussed in Section III. Also, the hypothesis is based on the intuition that the wider the range of the execution times is, the bigger the standard deviation value is.

**Hypothesis 2 (H2):** The average variation rate of the relative error of the hybrid model will positively correlate with the error rate of the classification model.

This hypothesis is drawn as the hybrid model is attributed from the classification. In other words, we hypothesize that if the error rates are high in the classification model, accordingly, the rates could contribute to the hybrid model.

To test these two hypotheses, we conduct correlational analyses among the four potential factors.
Table V exhibits the results of the correlational analyses across the sixteen simulation apps. To calculate the coefficients, we use the \texttt{cor()} function available in R.

From the results, we confirmed that H1 was statistically supported. More specifically, the correlation coefficient between the standard deviation of the run time and the classification model was 0.77, which was “high.” This means that there is a positive, strong correlation between these two factors. As shown in Fig. 5, we could visually confirm that there exists a strong linear relationship between the two as well.

Subsequently, we confirmed the validity of H2. That is, the correlational coefficient calculated between the average variation rate of the relative error of the hybrid model and the error rate of the classification model was 0.51, which was “medium.” This result says that the two factors positively correlate with each other. As depicted in Fig. 6, we could view that there is a significant linear relationship between the values of the two attributes.

Nevertheless, we did not see that there was little correlation between the remaining factors that we considered. The standard deviation of simulation run times has a significant influence on the classification and hybrid models, but the influence was low on the regression model, as indicated by the other numbers in Table V. Thus, more research should be conducted to investigate whether there are other factors that can significantly affect the performance of the proposed models.

Here is the summary of our findings. The classification and hybrid models worked effectively with the simulation apps with typically low standard deviation in their run times. That said, further research should be conducted on the regression model to see what other factors can affect its performance. Once the factors are identified, we expect that the machine learning models could be applied more effectively based on the characteristics of the simulation apps.

VI. CONCLUSION AND FUTURE WORK

This manuscript proposes EXTES—an execution time estimation scheme—for more efficient CSE simulation, especially conducted on a web-based platform named EDISON. We built a machine learning model of classification and regression, and also proposed a hybrid
model encompassing both the classification and regression techniques. Our models were trained and validated based on the provenance data from various types of simulation apps provided by EDISON. We estimated the execution time for a given simulation based on its input parameters without using any complex profiling information. As a result, our model showed high performance with an average of 73% accuracy for time estimation.

Although we faced a variety of challenges such as low-quality EDISON simulation run time data, black box testing for simulation apps with closed source code, lack of provenance data, and limited number of input parameters for simulation apps, we were still able to propose a run time estimation algorithm, demonstrating fairly high accuracy results. If we had a sufficient amount of provenance data and additionally available input parameter attributes, the accuracy of the proposed models for each of the simulation apps would be higher.

In addition, we expect that the efficiency of CSE simulations can be further enhanced in various ways using our proposed models. Benefits would include but not be limited to an increase in the convenience of simulation users, reduction of unnecessary resource consumption, and better chance of designing an optimal research plan that could save time and benefit the users and system administrators alike.

We see that future research will proceed in two directions. The first is to enhance the accuracy of our models without acquiring any more data. Section V.B. of this article describes factors that affect the error rate of classification and the relative error rate of the hybrid model. Based on these findings, those who want to expand this research can investigate more ways to improve our hybrid model by increasing the accuracy of the classification. It would also be interesting to examine how to improve the accuracy of regression after classification. We expect that exploring more machine learning models will bring new perspectives for boosting the accuracy of our models.

Another direction is designing and developing more useful tools for CSE simulation users and platform administrators. An optimal simulation job scheduler that calculates the best possible schedule based on the estimated execution time could be a valuable tool for both CSE simulation users as well as administrators. In addition to providing execution time forecasts, it would be an interesting research topic to develop a data analytics platform on the simulation provenance data. Future users can be guided to better conduct their simulations based on the knowledge obtained from the analytics.

ACKNOWLEDGMENT

The authors thank anonymous reviewers for their insightful comments and KISTI for providing the EDISON simulation data out of courtesy.

REFERENCES


