IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 13, 2020, accepted November 26, 2020, date of publication December 4, 2020,
date of current version December 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3042596

CLUTCH: A Clustering-Driven Runtime Estimation Scheme for Scientific Simulations

YOUNG-KYOON SUH[1], (Member, IEEE), SEOUNGHYEON KIM[1,2], AND JEEYOUNG KIM[1]
[1]School of Computer Science and Engineering, Kyungpook National University, Daegu 41566, Republic of Korea
[2]Samsung Research, Seoul 06765, South Korea

Corresponding author: Jeeyoung Kim (jeeyoungkim@knu.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Education through the Basic Science Research Program under Grant NRF-2018R1A6A1A03025109.

ABSTRACT Efficient scheduling among simultaneous simulation jobs is of critical importance in the allocation of limited computing and I/O resources. The difficulty of predicting when a job is completed can cause nontrivial problems for system administrators and users e.g., squandered resources, long waiting times, and simulation plan delays. To alleviate these problems, we propose a novel simulation runtime estimation scheme termed *CLUTCH*, which employs a well-orchestrated *ensemble* of clustering, classification, and regression techniques. The proposed scheme trains a runtime estimation model through a series of steps: (*i*) grouping past simulation provenance records by clustering, (*ii*) labeling each of the grouped records by classification, and (*iii*) performing regression on the execution times in each group. Given a simulation and its external arguments, the trained model predicts the simulation's runtime with high accuracy in a *black box* fashion, using only basic external arguments without needing extra information. We additionally propose two optimization algorithms which significantly reduce training overhead without sacrificing estimation quality. In the experiment with real datasets, our model achieved approximately a 14.2% growth in estimation accuracy, compared to the most recent state-of-the-art method; with our optimizations applied, the model was trained 16 times faster while still retaining accuracy.

INDEX TERMS Simulation runtime estimation, ensemble machine learning, pre-processing, simulation provenance, clustering, classification, regression, random forest, K-means.

I. INTRODUCTION

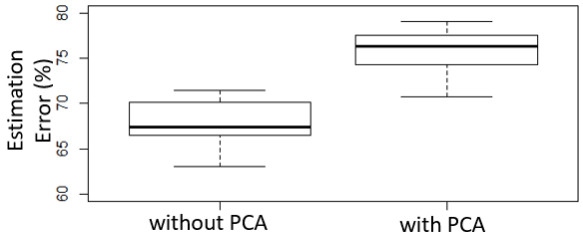
Runtime estimation has long been an important task for black box-based online simulation platform services [1]. The main concerns are that often many simulations accompany high-performance computing (HPC) and storage resources which accordingly require very high execution cost in time, sometimes reaching up to months. Such *long* execution times can lead to a variety of issues, such as (i) leaving users to sit and wait with no information of when their simulation will end; (ii) unexpectedly delaying simulation schedules; and (iii) wasting limited online simulation resources, occasionally caused by an infinite loop initiated by a wrong combination of simulation input values.

To address these aforementioned concerns, this article proposes a **CLU**s**T**ering-based s**CH**eme for estimating simulation execution time, which we call *CLUTCH*. The key idea of the proposed scheme is the application of an *ensemble* of clustering, regression, and classification techniques, rather than relying on a single prediction model. The model is accompanied by *two optimization techniques*, first, determining the optimal pre-processing permutation and second, finding the best number of clusters k in an automated fashion; the proposed optimization methods are capable of significantly reducing the training overhead while retaining the same estimation quality.

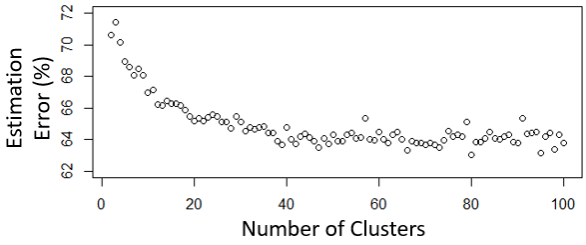
With *CLUTCH*, an opportunity for online simulation platform users to estimate their simulation times *a priori* will be offered, allowing the users to adjust their simulation schedules accordingly; further, the platform administrators will be able to develop smart schedulers to improve simulation job throughput [2], [3]. Additional technical details are discussed in Section IV.

Our *CLUTCH* scheme deals with two major issues which often affect the performance of runtime estimation. The first issue is *data pre-processing* a well-known and challenging problem which must be addressed before conducting full-scale analysis and model training. Common methods of

The associate editor coordinating the review of this manuscript and approving it for publication was Wentao Fan.



(a) Influence of the use of PCA on estimation error rate



(b) Estimation errors with increasing clusters

FIGURE 1. A motivating example of the importance of a well-chosen pre-processing method regarding estimation quality.

pre-processing include the location and elimination of outliers in the training data, or performance of normalization and elimination of duplicate data. Many other options exist, as well. We argue, however, that arbitrary data pre-processing without careful preliminary analysis may negatively impact the performance of a trained model.

Figure 1 clearly demonstrates that choosing a proper pre-processing method and a desirable hyperparameter value is critical for good performance. For a simulation program (named 2D_Comp_P) available on an online simulation platform, Figure 1(a) compares the simulation time estimation performance with and without Principal Component Analysis (PCA) [4], [5]. Applying a pre-processing permutation without PCA, the worst case prediction error rate on the program's data is 71.47%, with an average of 67.70%. In contrast, using a permutation with PCA yields an average error rate of 75.72% and a best case of 70.73%. These preliminary results emphasize that applying a pre-processing permutation without PCA to the simulation data can contribute to better prediction.

The second issue addressed by the proposed model is finding the optimal number of clusters k, in data clustering, which is a classic but still challenging problem as reported in many previous works [6], [7]. A rich body of existing literature proposes a rule of thumb to determine an optimal k value in k-means clustering variations.

Figure 1(b) plots the error rates of a simulation time estimation model employing k-means as the value of k increases. As illustrated, the estimation performance yields a minimum error rate of 63.01% and worst case error rate of 71.45%; the difference is a significant 8%. This demonstrates the importance of determining an optimal k value for better improving estimation quality.

The proposed optimization method in this article resolves the two following critical issues: yielding an optimal pre-processing permutation and finding the optimal number of clusters k for a given input dataset. With these optimizations, we achieve training times approximately 16 times faster and similar estimation performance. To the best of our knowledge, this is the *first* work to address these problems in runtime estimation.

Our contributions can be summarized as follows.

- A novel runtime estimation scheme is proposed, to apply an ensemble of clustering, regression, and classification techniques.
- Two optimization methods are presented for our scheme: first, to find the best pre-processing permutation, and second, to determine the optimal number of clusters.
- The performance of our models are evaluated on real datasets from an online simulation execution platform.
- It is shown through evaluation, that the proposed scheme achieves a relative growth of approximately 14% in accuracy, over the state-of-the-art method.

This article is organized as follows. In Section II related work has been reviewed and comparative analyses with our work have been performed. Section III includes a formulation of the simulation runtime estimation problem and a delineation of our approach. In Sections IV and V, we elaborate on the proposed scheme is elaborated and introduce two optimization techniques. Section VI conducts asymptotic analysis of the proposed scheme, and Section VII presents the evaluation results of the proposed scheme compared with state-of-the-art methods. Finally, Section VIII concludes our paper and suggests future research directions.

II. RELATED WORK

So far, there has been a rich body of existing literature in the area of estimating execution time. We have carefully and extensively examined the literature and narrowed down our focus to studies that we find relevant to this article. In this section, we describe in detail how our work differs from the previous studies in several aspects.

Table 1 summarizes major similarities and differences between our work and the existing studies. First, the examined papers concern various application domains: load sharing facility (LSF) [8], parallel program [9], [11], [25], cloud [10], [29], HPC [2], [14], [15], [19], location-based services [20]–[23], databases [26]–[28], big data applications [29], [30], and scientific workloads [12], [13], [16]–[19]. The runtime estimation problem addressed in this article applies to the scientific workloads domain.

Second, the target of estimation is slightly different across the studies we have investigated. Most of the existing works [2], [9]–[18], [29], [30] aim to predict the *runtime*. Some studies [2], [8] attempt to estimate the *memory usage*. A group of papers [20]–[23] proposes the estimation models of *arrival time* for intelligent transportation services.

TABLE 1. Qualitative comparison of our work with related existing studies.

Relevant Work	Problem Domain	Estimation Target	Estimation Tool	Hardware Parameters	Code Availability
Taghavi et al. [8]	LSF	Memory usage	Machine learning (linear regression)	Available	No
Park et al. [9]	Parallel Program	Runtime	Machine learning (support vector regression)	Available	No
Pham et al. [10]	Cloud	Runtime	Machine learning (random forest)	Available	No
Bhimani et al. [11]	Parallel Program	Runtime	Machine learning (linear regression) + Analytical model (stochastic markov model)	Available	No
Tanash et al. [2]	HPC	Runtime, memory usage	Machine learning (decision tree)	Available	No
Hilman et al. [12]	Scientific workload	Runtime	Machine learning (k-nearest neighbor), deep learning (long short-term memory networks)	Available	No
Nadeem et al. [13]	Scientific workload	Runtime	Machine learning	Available	No
Wang et al. [14]	HPC	Runtime	Machine learning (BAYES classifier), deep learning(RBF Network)	Available	No
Naghshnejad et al. [15]	HPC	Runtime	Machine learning(Adaptive Linear Regression) + Analytical model (Kalman filter)	N/A	No
Pumma et al. [16]	Scientific workload	Runtime	Machine learning (decision tree, ABC algorithm)	Available	No
Tyryshkina et al. [17]	Scientific workload	Runtime	Machine learning (random forest)	Available	Yes
Kim et al. [18] (our earlier work)	Scientific workload	Runtime	Machine learning (random forest, k-nearest neighbor)	N/A	Yes
Malakar et al. [19]	Scientific workload	Runtime	Machine learning	N/A	No
Li et al. [20]	Location-based Services	Travel time	Deep learning (deep residual networks)	N/A	No
Wang et al. [21]	Location-based Services	Travel time	Deep learning (multi-layer perceptron, long short-term memory networks	N/A	No
Fu et al. [22]	Location-based Services	Travel time	Deep learning (graph attention network, multi-layer perceptron)	N/A	No
Hong et al. [23]	Location-based Services	Travel time	Deep learning (graph neural network, convolutional neural network)	N/A	No
Heo et al. [24]	GPU program	Worst-case execution time	Deep learning (deep neural networks)	Available	No
Reder et al. [25]	Parallel program	Worst-case execution time	Machine learning	N/A	No
Sabah et al. [26], [27]	Relational database	Query time	Analytical model	N/A	No
Chu et al. [28]	Graph database	Query time	Deep learning (long short-term memory networks)	N/A	No
Ardagna et al. [29]	Cloud/Big data application	Runtime	Analytical model (queuing networks)	Available	No
Popescu et al. [30]	Big data application	Runtime	Machine learning (multivariate linear regression)	N/A	No
This paper (*CLUTCH*)	Scientific workload	Runtime	Machine learning (random forest, k-means)	N/A	Yes

Estimating the worst-case *execution time* is also discussed in some other works [24], [25]. Besides, there are some works [27], [28] concerning *query time estimation* in the database context. Our work heavily focuses on estimating *simulation runtime* with high accuracy.

Third, tools used for analysis differed across these studies. Some of these [26], [27], [29] developed (pure) analytical models and assess the validity of the model. Many of the existing studies [20]–[24], [28] build neural-net based learning models and utilize them for deriving estimated time; many others [2], [10], [16]–[19], [25], [30] use tree and linear regression based machine learning models. Some other works [11], [12], [14], [15] use hybrid methods combining these tools—analytical model, machine learning, and deep learning. In this article, we use what we consider the two most relevant of these works [16], [17] for performance comparison. In particular, these studies apply machine learning, but take a different approach. For instance, Pumma *et al.* [16] organize a given workload into a decision tree and estimate the runtime using the Artificial Bee Colony (ABC) algorithm [31]. Tyryshkina *et al.* [17] explore various regression models for runtime estimation and conclude that the random forest offers the best performance. Considering the fact that machine and deep learning appear de-facto standard tools for time estimation, and that the number of simulation provenance records of each scientific simulation program is not sufficient to apply deep learning, it is natural to employ the random forest model to solve our problem.

Fourth, the availability of hardware-specific parameter data differs from the existing works. The algorithms proposed in some of the studies [2], [12]–[14], [16], [17], [24], [29] require hardware information that may enhance the quality of the runtime estimation, while those proposed in some other studies [15], [18] require no such information. The former

is known as the *white box*, and the latter, the *black box*. Our runtime estimation approach takes the black box fashion, which is much simpler and more lightweight than the white box in which heavy resource monitoring is necessary.

Fifth, the majority of the existing work raises a "reproducibility" issue. Their source code is not open to public. Some other work, however, was reproducible; for instance, Tyryshkina *et al.*'s work [17] and our earlier work [18] made their code available, and Pumma's algorithm [16] was easily reimplemented. In Section VII, we will compare the performance of our proposed scheme with that of these three previous methods on the same simulation datasets.

Last but not least, the previous run-time estimation method [18], named *EXTES*, is an ensemble of classification and regression. Here are the major distinctions between *EXTES*, and the proposed scheme, *CLUTCH*. Although both are ensemble methods, our new scheme applies "clustering" to substitute for the classification of the previous work - that is, *CLUTCH* applies a clustering technique to group similar simulation provenance data, whereas *EXTES* uses a very simple method of dividing execution time. Moreover, determining the optimal number of clusters among the data turns out to be highly effective to the new scheme. *CLUTCH* succeeds in further improving performance with the inclusion of that optimization. As a result, the proposed scheme results in superior estimation quality to that of previous work by a large margin.

III. PRELIMINARIES

This section provides some preliminaries necessary to understand our paper. We introduce terminology, notation, and definitions; formulate our simulation time prediction problem; and then give a brief description of our approach.

A. TERMINOLOGY

Below, we explain several terms used throughout this article.

- *Simulation program*: A computer program running on an online scientific workload execution platform. It receives from its users one or more input parameters (defined below) and performs scientific computation.
- *Simulation input parameter(s)*: A set of values provided as input for a specific simulation program.
- *Simulation instance*: A simulation activity running with user-specified input parameters. A simulation instance consists of a set of input parameters and its corresponding runtime, as defined below.
- *Simulation runtime*: The end-to-end time of a given simulation instance. Note that the runtime significantly varies not only across different simulation programs but also with different simulation input parameters. Also, even when the same input parameters are entered into the same simulation program, their runtimes may vary.
- *Simulation provenance*: A "bag" of simulation instances. The entire provenance is treated as an input dataset for constructing a runtime estimation model.

TABLE 2. Notations and descriptions.

Notation	Descriptions
P	# of input parameters of a simulation program
N	# of simulation instances of that program
X	A bag of simulation input parameter data
Y	A bag of simulation runtime data
M	A model for simulation runtime estimation

B. NOTATIONS AND DEFINITIONS

Table 2 lists the notations used throughout the paper.

- *Input parameters:* Each simulation input datum, $x_i \in X$, contains P input parameters, which are expressed as $[a_1, \ldots, a_P]$, where an input parameter, a_j, corresponds to the j-th parameter of the P input parameters. The datum, x_i, can then be represented by the following one-dimension tensor: $x_i = [a_{i,1}, \ldots, a_{i,P}]$, where $1 \leq i \leq N$.
- *Simulation runtime:* Y contains N simulation runtimes. Y can be represented by $Y = \{y_1, \ldots, y_N\}$. A runtime y_i is driven from an input datum x_i. Our study aims to generate a model M to predict runtime for a given set of the P input parameters on a simulation program. The runtime estimated through M is defined as $\hat{y}$.
- *Data to predict:* After training the model M based on the input parameters and runtime data, we predict the runtime for a given simulation instance with the P input parameters. For that input parameter datum for the new instance, x_j, which can later be included in X, the predicted runtime is defined as $\hat{y}_j$, where $j \geq N + 1$.

C. PROBLEM FORMULATION

The goal of this article is for each simulation program to build and utilize the best model to estimate the runtime $\hat{y}_j$ on a given simulation instance, x_j, with the P input parameters. In the following, we formulate our research problem.

1) MODEL CONSTRUCTION

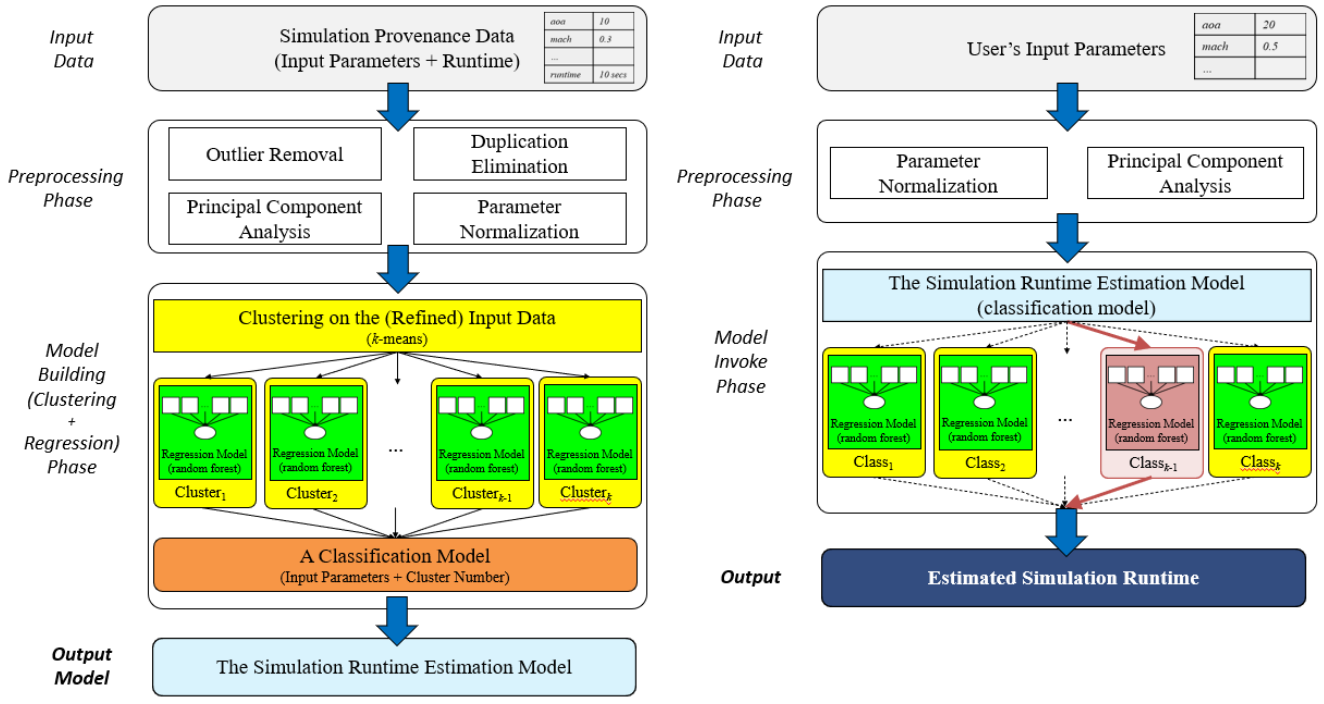
For all elements in X, our goal is to train M to minimize $\frac{\left(\sum_{i=1}^{N} |\hat{y}_i - y_i|\right)}{|Y|}$. In other words, we want to find M, that can most reduce the sum of errors between the estimated $\hat{y}_i$ and the captured (true) time, y_i.

2) RUNTIME ESTIMATION

This concerns reporting the simulation runtime estimated by the developed model M, an ensemble of clustering, regression, and classification methods. This problem is much simpler than the former model construction problem: given a new simulation instance, x_j, the goal is to estimate and report the simulation time of x_j: that is, $\hat{y}_j = M(x_j)$.

D. OUR APPROACH

In this study we take three key approaches for simulation runtime estimation. Below, we describe the reasons for and benefits of each selected approach in detail.



(a) Machine learning model training process

(b) Execution runtime estimation process

FIGURE 2. An overview of the proposed *CLUTCH* scheme: for a simulation program, this figure illustrates training a runtime simulation model and later invoking the model to produce the runtime predicted by the model on a user's input parameters.

The first approach relates to *input values for estimation*. Many existing works [2], [12]–[14], [16], [17] take advantage of hardware-specific parameters that may be useful for runtime estimation as mentioned in Section II. However, the measured hardware parameters are *not* always available across all platforms. Thus, for a certain environment in which such parameters do not exist, the existing techniques cannot be directly applied. Even if applied, estimated quality may decrease. To address this concern, we estimate simulation runtime based on only input parameters that are typically stored in almost all platforms, without needing hardware-specific parameters. By doing so, we can obtain two nice benefits: (1) *saving efforts and resources* for hardware resource monitoring and (2) *applicability* to a platform or an application that cannot support such hardware monitoring.

The second approach involves the *estimation tool*. Many of the existing works exhibited in Table 2 rely on machine learning as a tool. Their machine learning models worked well for their estimation. Considering their success, it is totally appropriate to leverage machine learning techniques to solve our time estimation problem.

The third and last approach concerns a *combination of prediction methods*. In this article we present a novel "ensemble" of clustering, regression, and classification to construct a runtime estimation model. Unlike previous works [13], [18] which take a similar approach, the biggest difference in our work is to apply *clustering* to the input data in the early training process, resulting in estimation quality enhancement.

IV. PROPOSED SCHEME

This section explains in detail our scheme estimating runtime on specified input parameters of a simulation program.

A. OVERVIEW

Figure 2 illustrates an overall flow of our proposed scheme, *CLUTCH*, comprising two processes: developing a runtime estimation model (Figure 2(a)) and predicting the runtime via the model (Figure 2(b)).

Figure 2(a) overviews the process of training the model. In the training process, the proposed scheme builds three models: *clustering* (colored in yellow), *regression* (in green), and *classification* (in orange). First, for a simulation program, its simulation provenance data—input parameters and their runtimes—are pre-processed prior to actual model training. Next, the refined provenance data are grouped together into k clusters. In other words, the provenance data are clustered in accordance with the characteristics of the internal simulation computation. Then, for each of the k clusters, their respective regression model is built based on the simulation input parameters of the cluster. The training is finalized by labeling the clusters with their respective numbers and input parameters. In this way, the model-building process yields the simulation runtime estimation model (in sky blue) for a given simulation program.

Figure 2(b) pictorially abstracts how the proposed *CLUTCH* scheme reports the final estimated runtime via

the developed model from that program. *CLUTCH* accepts user-specified simulation input parameters of the same simulation program and pre-processes the input data through parameter normalization and PCA. Then, the trained model is utilized to identify which cluster is closest to the user input data and then pull out the regression model associated with the cluster. Finally, our scheme predicts and outputs the runtime on the input data via the regression model. (The red arrows and boxes, for instance, represent the path to choose a desired class of the estimation model and calculate the predicted runtime.) Note that both processes must first go through the respective pre-processing phases, which we find critical in enhancing the quality of the runtime estimation.

B. DETAILED SIMULATION RUNTIME ESTIMATION

This section elaborates on developing a runtime estimation model. Note that since the parameter information for each simulation program is different, the estimation model must be separately built and trained. Nevertheless, the overall methodology is applied across all programs. Next we discuss how pre-processing is carried out on the data.

1) PRE-PROCESSING PERMUTATION ON SIMULATION RUNTIME DATA

Figure 3 shows the error variation for a simulation program as the number of applied pre-processing methods increases. It is clear that the error rate increases when more than three methods are applied. More importantly, Figure 3 implies that, regarding estimation time performance, it is critical to determine the best pre-processing permutation among the methods.

To yield the optimal permutation, we consider the following four pre-processing methods: (i) *parameter normalization*, (ii) *redundant parameter instance elimination*, (iii) *dimension reduction through PCA*, and (iv) *outlier elimination*. The first method indicates rescaling the range of simulation parameter data between 0 and 1. The second method is to integrate redundant provenance data with the same parameter values into one. Note that even if the same parameters are entered into the same simulation program, the actual execution time may vary due to time variability and other external factors. In our study, the runtimes of each unique set of parameters are averaged (investigation into better strategies is needed). The third method is to apply PCA [4], [5]. Since scientific simulations are based on a number of input parameters, PCA is a reasonable and effective option. The last pre-processing method is to remove outliers. When looking into the runtime distribution of the programs, we find that most of the runtimes are concentrated near a certain point. (This point is close to the average of the runtimes.) The further away from that point, the fewer data there is. Hence, we can infer that the outlying data produce more errors and thus must be eliminated.

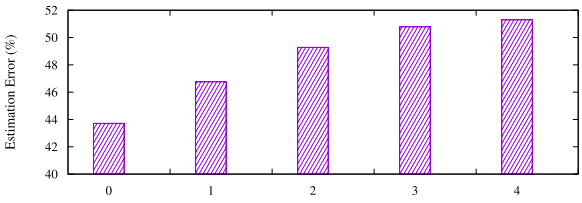


FIGURE 3. Error variation according to number of pre-processing methods applied.

Equation 1 shows the runtime range (RT) of simulation provenance data that remains after removing outliers:

$$RT : avg - c \times sd \leq t \leq avg + c \times sd \tag{1}$$

where t: a simulation runtime, avg: average of runtime, sd: standard deviation of runtime, and c: a constant in $\left[0, \frac{avg}{sd}\right]$.

Data that have a greater distance than the standard deviation of the runtime weighted by a certain constant factor c from the mean of the runtime is treated as an anomaly and removed. There are two criteria for determining c. The first criterion is to find an *elbow*, the point where a lot of data is sharply removed. Our study defines the elbow as the point at which at least 1% of the data is removed. That said, a large amount of data may be removed along with the elbow. Therefore, the second criterion is to limit the maximum amount of data that can be discarded. If the elbow is not found, we remove up to 5% of the total data and treat the retained data as valid.

The aforementioned four pre-processing methods need to be applied in a certain sequence to improve estimation quality. We thus seek the best permutation of these methods by making estimations and comparing quality for each permutation. The number of pre-processing permutations is derived as shown in Equation 2. Given the four methods, we can come up with a total of 65 permutations by substituting 4 for m in the following Equation 2:

$$U = \sum_{i=0}^{m} \binom{m}{i} \times i! \tag{2}$$

where U: the number of pre-processing permutations, m: the number of pre-processing methods.

2) ALGORITHM FOR DEVELOPING A RUNTIME ESTIMATION MODEL

We now elaborate on the training algorithm. Algorithm 1 illustrates how the model is constructed. The algorithm takes as input the past parameter data of a simulation program. We first obtain an initial model M via regression (Line 1). That model is equivalent to the regression model *covering* a single cluster of the instances. M holds the best model so far. In turn, the algorithm loops over a range from two to one hundred clusters (Lines 2–12). Specifically, it performs clustering and stores the result into C_k of M_c indicating a candidate (Line 3). Next, for each cluster in C_k, its regression model (rg) is explored, and its error is summed as e (Lines 4–7).

Algorithm 1 A Runtime Estimation Model

input : $inD \leftarrow$ A program's simulation provenance data
output: $M \leftarrow$ The program's runtime estimation model
1 $M \leftarrow$ getRgMdl(inD) ; // M: The best model
2 **for** $k \leftarrow 2$ **to** 100 **do** // k: # of clusters
3 $M_c.C_k = \text{getClMdl}(inD, k)$; // M_c: A candidate in consideration
4 **foreach** $c_k^i \in C_k$ **do** // c_k^i: i-th cluster in C_k
5 $M_c.C_k.rg_k^i = \text{getRgMdl}(c_k^i.inD)$;
6 $e \mathrel{+}= \text{computeError}(M_{cand}.C_k.rg_k^i)$;
7 **end**
8 $M_c.err_k \leftarrow \frac{e}{\sum_{i=1}^{k} |c_k^i|}$;
9 **if** $M_c.err_k < M.err$ **then**
10 $M \leftarrow M_c$; // Set M_c to M.
11 **end**
12 **end**
13 **return** M

We then compute a relative error ($M_c.err_k$) by averaging e over the sum of the non-redundant simulation instances included in each cluster (Line 8). If the calculated error is fewer then the error of M ($M.err$), then the running candidate (M_c) is updated to M, which becomes the new best model (Lines 9-11). After the range is exhaustively examined, the training ends and yields M as the final model (Line 13).

3) REPORTING ESTIMATED SIMULATION RUNTIME

After this training process, as already shown in Figure 2(b), the proposed scheme accepts and pre-processes given simulation input data and then identifies a particular cluster through an additional classification model [32]. This classification model is trained to associate a cluster with its input simulations as illustrated in Figure 2(a). Note that in the real environment, a runtime value is not given as input. That is why we need the classification model to use only simulation input parameters in order to determine the best matching cluster. Our scheme finally reports the estimated runtime through the regression model of that chosen cluster.

4) UTILIZED MACHINE LEARNING MODELS

Regarding machine learning tools, we use random forest [33] for both regression and classification, as it has been proven to show its excellent performance in many existing works [10], [17], [18], [32]. When it comes to clustering, we use k-means [34], and the reason will be detailed in Lemma 1 in Section VI.

V. OPTIMIZATION

Our *CLUTCH* scheme proposed in Section IV can fall victim to two potential problems. First, considering all possible permutations of the four pre-processing methods may take an unacceptably long time. Second, as seen in Algorithm 1, a large number of clusters under consideration may significantly impact the model's training time. To mitigate these problems, we introduce two optimization techniques.

A. PRUNING PRE-PROCESSING PERMUTATIONS

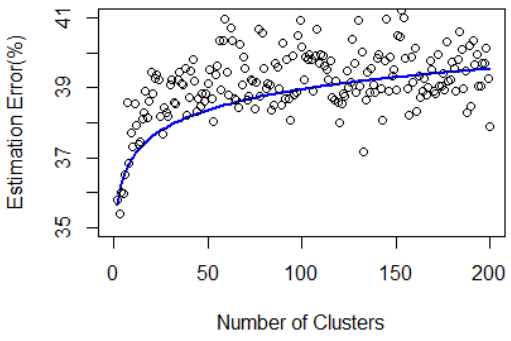
Putting the four pre-processing options on the table, we consider 65 permutations by default as simply computed in Equation 2. This implies that for each simulation program, the training and verification processes of the model should be carried out 65 times in total. Considering a number of simulation programs, it is practically infeasible to apply all permutations to build a simulation execution time estimation model associated with each of the programs. Thus, it is essential to avoid excessive training time by applying early pruning to as many uncompromising permutations as possible. This section delineates how we *can* drastically reduce the total number of permutations in consideration to eleven, without impacting the overall estimation accuracy.

The first consideration is the *sequence* in which the pre-processing methods are applied. If the ordering of the pre-processing methods yields little difference in simulation execution time as well as minimal difference in the predicted quality, then we can save training time by using a fixed sequence. Based on this assumption, we explored cases in which the same results were obtained regardless of the sequence of the pre-processing methods. Consequently, some combinations, such as 1) normalization and duplicate elimination and 2) normalization and outlier removal, did not matter to their internal orders.

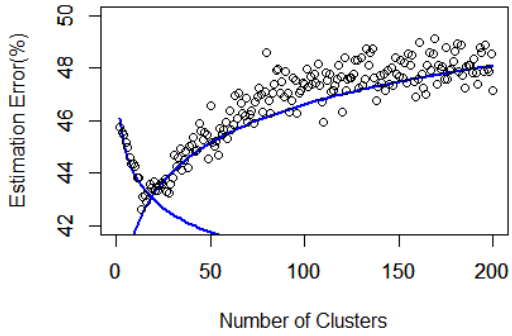
Further analysis revealed that, on the contrary, the order of application of normalization and PCA has a significant impact on estimation quality. More specifically, the permutations with PCA after normalization showed an error of 50.16% on average—a difference of about 5% from the opposite order's 55.10% error. It is therefore advantageous to run PCA after normalization. Based on these results, the following order has been determined as a permutation with high accuracy: **outlier removal → duplicate elimination → normalization → PCA**.

By deriving the above sequence, the problem of calculating the number of permutations to be considered is narrowed down to examining the applicability of each technique; thus Equation 2 can be updated as simply 2^N (N: number of pre-processing methods). Given the four pre-processing options currently available, a total of 16 permutations can be considered.

Next, we analyzed whether applying the pre-processing methods indeed affected estimation quality. Recall that Figure 1(a) shows only pre-processing methods that adversely affect the runtime data of a particular simulation program. The example demonstrates that including more methods does *not* always guarantee better performance. We also found that as the number of methods to consider increased, the runtime estimation accuracy tended to drop in many cases (again, refer to Figure 3). In addition, the best predictive quality for a simulation program was obtained when permutations with two or fewer methods were applied, while the worst quality



(a) Simulation program, A



(b) Simulation program, B

FIGURE 4. Error rate variation of cluster increase on two simulation programs.

was often shown when three or more methods were applied. We therefore concluded that it is optimal to choose two or fewer appropriate pre-processing methods. Based on these results, approximately five-sixths of the original permutations were pruned, resulting in only 11 permutations.

B. FINDING THE OPTIMAL k CLUSTERS

To address the model's second issue, we herein propose how to efficiently find the optimal number of clusters, k.

We conducted a preliminary analysis on our simulation runtime data (exhibited in Table 3). Interestingly, we found three underlying global trends present across the data. Data following the first trend displayed an error rate that rose with the number of clusters, as shown in Figure 4(a). The second trend showed the error rate fall then rise at a certain point, as the number of clusters increased, and is shown in Figure 4(b). Data of the third type showed no distinctive form.

In the first two trends, we observe that as the number of clusters increased, the error rate gradually increased as well. This common property reveals that the variation of the error rate is closely akin to a logarithmic function (represented by a blue line in Figure 4). More specifically, the first trend can be described as a log function with positive coefficients; the second trend can be described as a combination of two log functions where the first log function (to the left of the lowest error point) has negative coefficients, while the second (to the right of the point) has positive coefficients. The point at which

TABLE 3. Summary of the datasets used for our experiments.

CSE Area	Dataset (Simulation Program)	# of Records (Simulation Instances)	Runtime (sec)			
			Min.	Max.	Avg.	Std. Dev
CFD*	2D_Comp_P	9,241	2	2,060,879	35,086	93,479
	2D_Incomp_P	17,543	2	2,350,511	21,592	69,590
	KFLOW_4	652	2	105,182	8,147	12,956
	KFLOW_5	173	3	246,826	20,864	36,219
	SNUFOAM_ShipRes	1,123	360	360,467	64,822	57,540
CHEM*	dmd_pol	228	3	249	94	35
	eklgcmc2	82	5,896	600,881	72,753	128,913
	mc_nvt	1,750	2	1,512	17	53
CMED*	PKsimEV	417	12	846,496	2,092	41,451
	Single_Cell_Electr.	1,323	2	16,681	59	544
NANO*	acuteSTMtip	5,752	3	4,005	78	225
	BAND_DOSLab	2,192	2	65,375	348	1,709
	coulombdart	6,965	3	72	5	1
	gravityslingshot	15,818	7	34,115	4,111	4,707
	LCAODFTLab	1,577	2	609,942	15,523	56,699
	PhaseDiagramSW	1,586	3	50,689	697	1,260
	pianostring	1,001	27	15,379	1,246	1,381
	roundSTMtip	5,936	7	3,237	64	139
	WaveSimulation	22,407	3	85,441	389	2,230

(*CFD: Computational Fluid Dynamics, CHEM: Computational Chemistry, CMED: Computational Medicine, and NANO: Nano Physics)

the second trend has the lowest error is the intersection of the two log functions.

Figure 5 visualizes finding the best k for our problem. In the figure, the x-axis indicates the number of clusters, k, increasing to the right, and the y-axis represents the error rate associated with each k value.

Algorithm 2 describes the process of determining the best k as illustrated in Figure 5. Step 1 calculates the estimated error of the sampled k for regression (Lines 1-6). We draw more samples when the k value is small, because as k increases, the time required for clustering also increases. Thus, k starts at 2 and increases by a factor of 1.6, being rounded up to the nearest integer. The total number of sampled k values is 13.

Step 2 performs logarithmic regression based on calculations of cluster counts and errors for a total of 13 samples (Lines 8-14). First of all, the fit for Case 1, which represents the first trend, is the same as the traditional logarithmic regression. This can result in an expression of the form $y = a \cdot \ln(x) + b$, which accounts for the entire sample. The fit of Case 2, which represents the second trend, performs two logarithmic regressions: the first fit, $y = a \cdot \ln(x) + b$, on the left, and the second fit, $y = c \cdot \ln(x) + d$, on the right. Considering that (i) there is a total of 13 sampled k values and (ii) at least two values are required for a fit, the total number of possible t values (meaning the points yielding the smallest error) is 10. Each of the t values produces its respective error. As the fit with the least error most explains the sample, the subsequent step uses that fit.

Step 3 verifies whether the Case 2 fit satisfies the condition of "$a < 0$ && $c > 0$" (Lines 17-23). If the condition is true, then the model follows the second trend, and we can determine the optimal k from the logarithmic functions' point of intersection. Else, the model may follow the first trend or third trend.

In Case 1, if a is positive, then the model represents the first trend. Hence, the optimal k is determined as the smallest number of clusters, which is 2. If a is 0, on the contrary, the

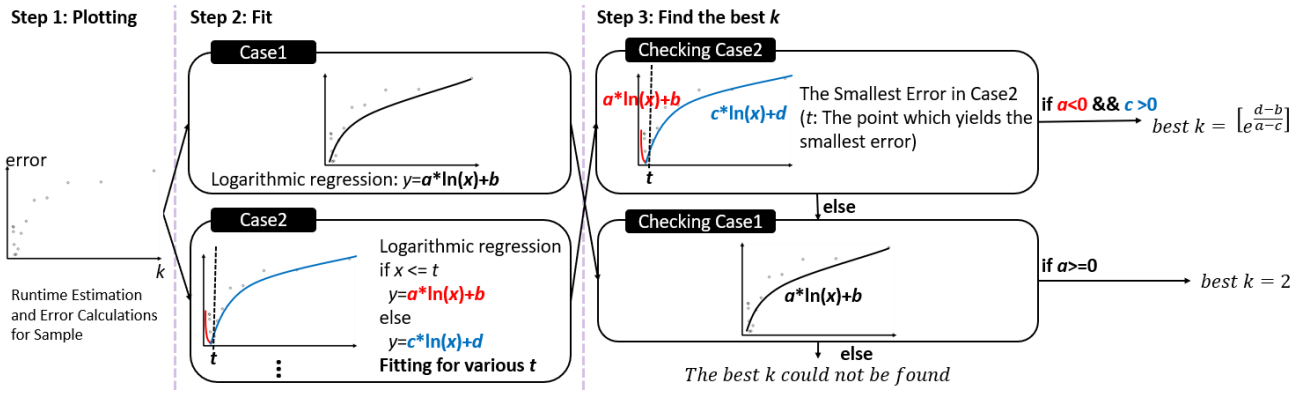


FIGURE 5. Visual example of the process to automatically determine k.

Algorithm 2 Finding the Best Number of Clusters, k

input : Simulation parameters and runtimes
output: The optimal cluster count, k

// Step 1: Generate Samples for case checking
for $i \leftarrow 0$ **to** 12 **do**
  $k = \lfloor 2 \times 1.6^i \rfloor$;
  $x[i+1] = k$;
  $y[i+1] \leftarrow$ Calculate the error for k ;
end
// Step 2: Fitting for case checking
$lrCs1 \leftarrow$ Log regression ($y = a\ln(x) + b$) on Case 1 ;
for $i \leftarrow 2$ **to** 11 **do**
  $x_1 = x[1:i]$;
  $x_2 = x[1+i:13]$;
  // Compute log regression on Case 2
  $y = (x \in x_1)\ ?\ a\ln(x) + b : c\ln(x) + d$;
end
$lrCs2 \leftarrow$ Model with the smallest error for Case 2 ;
// Step 3: Fitting for case checking
if $lrCs2.a < 0$ && $lrCs2.c > 0$ **then**
  $k_{best} = \left[e^{\frac{d-b}{a-c}}\right]$;
else if $lrCs1.a \geq 0$ **then**
  $k_{best} = 2$;
else // Cannot find the best k
  $k_{best} \leftarrow$ Not available;
end
return k_{best}

error is *constant*. In this case, k is set to be 2 to save time on clustering. If the a value does not satisfy either of these two cases, we cannot find the best k; this case corresponds to the third trend. In such case, the entire range of k is examined in the same way as the scheme without optimization.

Note that a few more clusters around the chosen k are considered, as k may not always guarantee optimal performance.

VI. ANALYSIS

Here we show the asymptotic complexity of our scheme.

Lemma 1 (Clustering): The time complexity of clustering N simulation instances is $O(k \cdot N \cdot P)$, where k is the number of clusters and P is the number of simulation input parameters.

Proof: In our scheme, k-means is used as a tool for clustering. k simulation instances are initially chosen as center points. For $(N - k)$ simulation instances, their Euclidean instances from the k centroids are calculated to form k clusters. The calculation iterates over P parameters (a.k.a. P dimensional space) that each of the simulation instances has. Hence, the asymptotic complexity of our clustering is equal to $O(k \cdot N \cdot P)$. □

Remark 1: PAM [35] can be used as an alternative to the k-means used in the proof. However, its time complexity is known as $O(k(N-k)^2)$. This complexity is much less efficient than that of k-means. It may be possible to apply sampling-based clustering algorithms that are more rapid than PAM and k-means; however, the sampling-driven approach cannot be considered if N is insufficient.

Lemma 2 (Regression): The time complexity of the regression for N simulation instances is $O(T \cdot S \cdot N \log(N))$, where T is the number of trees and S is the number of attributes chosen by sampling.

Proof: In our scheme, random forest is used for regression. Suppose that we sample S parameters or attributes (from P defined in Lemma 1) of a simulation program. For each $s \in S$, its tree can be built within $O(N \log(N))$ time. Because the number of trees is T, the running time complexity of the regression is equal to $O(T \cdot S \cdot N \log(N))$. □

Lemma 3 (Classification): The time complexity of the classification for N instances is equal to Lemma 2.

Proof: We use random forest for classification as well. Thus, the time complexity is equal to that of the regression. □

Theorem 1 (The Proposed Scheme): For a simulation program, the training time complexity of its runtime estimation model is $O(P \cdot N \log(N))$, where N is the number of instances, and P, the number of simulation input parameters.

Proof: According to Lemma 1, running k-means for N simulation instances takes as much time as $O(k \cdot N \cdot P)$.

Lemma 2 indicates that the time taken for random forest is $O(T \cdot S \cdot N \log(N))$. The running time of performing classification is also $O(T \cdot S \cdot N \log(N))$. For the regression within each cluster, its running time is equal to $\sum_{j=1}^{k}(T \cdot S \cdot N_j \log(N_j))$, where N_j indicates the number of simulation instances in the j-th cluster created after k-means is performed.

Considering the three costs, the running time complexity for training the model is $O(\sum_{k=k_{min}}^{k_{max}}(k \cdot N \cdot P + T \cdot S \cdot N \log(N) + \sum_{j=1}^{k} T \cdot S \cdot N_j \log(N_j)))$, where k_{min} is the minimum value, and k_{max}, the maximum value of the range to find the best k. Since $\sum_{j=1}^{k} N_j \log(N_j) \leq N \log(N)$ and $S \leq P$, the equation can be modified to $O(\sum_{k=k_{min}}^{k_{max}}(k \cdot N \cdot P + T \cdot P \cdot N \log(N) + T \cdot P \cdot N \log(N)))$. By arithmetic sequence, this equation can be rewritten to $O(N \cdot P \cdot (\frac{k_{max}^2 + k_{max} - k_{min}^2 + k_{min}}{2}) + 2 \times (k_{max} - k_{min} + 1) \times (T \cdot P \cdot N \log(N)))$. Here all the terms involving k_{max}, k_{min}, and T can be simply treated as constants. Then, the continuing equation is further modified to $O(c_1 \cdot P \cdot N + c_2 \cdot P \cdot N \log(N))$, where c_1 and c_2 are some constants. Hence, the total training time is $O(P \cdot N \log(N))$. □

Remark 2: Theorem 1 emphasizes that our scheme asymptotically scales well with increasing simulation instances. The following section provides empirical evidence on the scalability of our scheme on real datasets.

VII. EXPERIMENT

We now present our evaluation results. The evaluation focuses on the estimation quality, effectiveness of optimization, and training time overhead of the proposed scheme.

A. ENVIRONMENT CONFIGURATIONS

We describe our experiment settings and explain and analyze the datasets used for evaluation.

1) EVALUATION ENVIRONMENT

Our proposed scheme *CLUTCH* was developed in R [36] and the source code is currently available in a GitHub repository.[1] The evaluation was performed on a Windows 10 Education OS server running on an Intel Xeon Gold 6126, 32 GB RAM, and a 1 TB M.2 NVMe SSD. We created Hyper-V VMs running Ubuntu 18.04 using 1 Core and 8 GB RAM allocated for evaluation.

2) DATASET

To make a fair performance comparison of our scheme with the state-of-the-art methods, we use the datasets presented in our prior paper [18]. The datasets, which are also available at the same Github repository, include runtime data collected on an online service platform [37] supporting various computational science and engineering (CSE) simulations. Details of the datasets used are exhibited in Table 3.

In the datasets, most records (or simulation instances) have a very long average runtime, reaching up to two million seconds. There are also quite a few abnormal cases where the measured runtimes are less than two seconds. It is safe to assume that running time below two seconds may have occurred due to a runtime error or being inadvertently stopped by the user; we, therefore, discard such records and exclude them from the data for training and evaluation.

3) EVALUATION METRICS

Establishing a good performance metric is essential for accurate assessment. In this section we discuss two possibilities for the metric and proceed with the reasonable choice between the two.

The first metric is the Mean Absolute Percentage Error (MAPE) [38] method, the most basic method that can be considered for comparison with other competitors. MAPE is defined below in Equation 3, where n is the number of records, A is actual runtimes, and F is predicted runtimes.

$$MAPE(\%) = \frac{100}{n} \sum_{t=1}^{n} \frac{|A_t - F_t|}{A_t} \tag{3}$$

Since the range of the time reported by the proposed scheme can be very wide (as exhibited in Table 3), there may be a large gap between the predicted time and the actual time. For instance, if the actual time is short and the predicted time is long, the error value becomes very large. Because MAPE averages each error, such large error values can distort the estimation results of other good performances; a method is therefore needed to eliminate this distortion.

A previous study [39] has shown that the shortcoming can be overcome by applying the Symmetric Mean Absolute Percentage Error (SMAPE), as shown in Equation 4.

$$SMAPE(\%) = \frac{100}{n} \sum_{t=1}^{n} \frac{|A_t - F_t|}{(A_t + F_t)/2} \tag{4}$$

For SMAPE, the difference between the predicted and the actual runtime is divided by the mean of the actual and predicted runtime, instead of the actual runtime, reducing distortion compared to MAPE. A certain idiosyncrasy of the SMAPE approach is that unlike most percentage-based evaluation methods producing errors up to 100%, in the worst case SMAPE may produce errors up to 200%.

Due to the fact that the range of errors is from 0% to 200%, and that the range of time predicted is vast, there still exists distortion on the overall results, so additional correction methods are needed. An estimation value that is far off from the average of the estimation results is considered to be an anomaly and the outlier removal techniques given in Section IV-B1 are applied. Through this process, we can obtain the estimation result with limited distortion.

B. EVALUATION RESULTS

In this section we discuss our evaluation results.

Using the datasets shown in Table 3, we compare the performance of our proposed scheme, *CLUTCH*, with two state-of-the-art works [16], [17]. Tyryshkina *et al.* [17] make their code available as exhibited in Table 3, so there is no

[1] https://github.com/knudeallab-papers/clutch

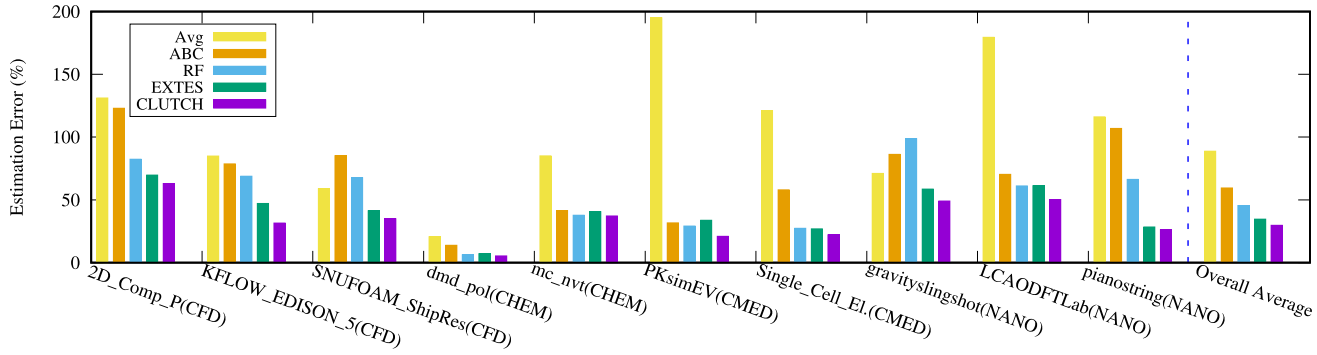


FIGURE 6. Evaluation results on estimation error across the different simulation programs.

problem with comparing their performance with our own. We have also reconstructed the code from Pumma *et al.*'s algorithm [16] to fit our data, and we compare that work with our *CLUTCH* scheme. As our earlier work [18] deals with the same data used in this study, it is extensively compared with the proposed work. We additionally include the comparison results of a baseline method, termed '*Avg*,' simply reporting an average simulation time of each program. One thing to note is that comparison with Naghshnejad *et al.*'s work [15], which can't be directly adapted to our proble, is not possible. This is because their algorithms utilize program submission time, resulting in unfair comparison with our work.

Figure 6 displays the comparison results of estimation error across simulations. In the figure, 'ABC' is the result of the previous work [16] on estimating runtime using the ABC algorithm. Again, we reconstructed their work for our comparison. 'RF' is the result of another previous work [17], whose authors compared various regression models and found that the random forest model yielded the best results; we therefore use the random forest for 'RF'. 'EXTES' is the result of our most recent work [18]. We previously proposed a hybrid model of classification and regression for runtime estimation. The results were selected from several representative programs in each CSE (computational science and engineering) domain, with the average for all the nineteen programs on the far right. (Because of lack of horizontal space, the rest of the programs can't be shown here although we do have their results.)

As shown in Figure 6, our proposed scheme, *CLUTCH*, near-consistently yielded the best performance among other state-of-the-art methods (as well as 'Avg'). More specifically, the average error rates for all simulations were 59.46% for ABC, 45.51% for RF, and 34.67% for EXTES, while *CLUTCH* (our scheme) showed a 29.75% error rate. This is a 14.19% increase in accuracy over EXTES, which was the best among the three competitors. The error rate of *CLUTCH* was as low as approximately 5% on the dmd_pol program (from the CHEM area in Table 3), in particular. These empirical results demonstrate the validity and effectiveness of the clustering-driven approach used in the proposed scheme.

C. OPTIMIZATION PERFORMANCE

In Section VII-B we have shown that the proposed *CLUTCH* method betters the recent methods [16]–[18]. Nevertheless, we are aware that the training process takes a long time, as mentioned previously. To address this concern, we introduced several optimization methods in Section V. Here, we present the evaluation results with optimization applied, to confirm the effectiveness of the proposed optimization techniques.

Figures 7 and 8 show the result of incrementally applying different optimization options of *CLUTCH* to the selected programs in Table 3. In the figures, 'CLU_v0' represents the vanilla version (with no optimization applied), as evaluated in Section VII-B. 'CLU_v1' represents the result of applying the method of finding the optimal k clusters (denoted as 'OPT1') to CLU_v0. 'CLU_v2' represents the result of applying the method of reducing pre-processing permutations (denoted as 'OPT2') to CLU_v0, and the last version, 'CLU_v4', represents the result of applying the two optimization options together to CLU_v0. On one hand, an earlier study [40] proposed an algorithm, termed *X-means*, to be able to quickly determine the optimal number of clusters (denoted as XM). Since XM can be used in place of OPT1, we add the result of replacing OPT1 with XM in CLU_v4, represented as 'CLU_v3'.

Figure 7 shows the uncompromising performance of OPT1 and OPT2. The difference in error rates among the variations of *CLUTCH* except CLU_v3 was in the range of 0.2%∼0.4% across the board. Considering the effect of random factors in the model such as random forest, the quality of estimation almost never varies. But when X-means [40] was applied by CLU_v3, the estimated quality is adversely affected. Hence, X-means *cannot* find the optimal number of clusters that we wanted to determine.

Figure 8 shows differences in training time among the different versions of *CLUTCH*. As illustrated, the proposed optimization methods contribute to a reduction of training time; more specifically, compared to CLU_v0, CLU_v1 is about 2.9 times faster, CLU_v2 is 5.3 times faster, CLU_v3 is 64.8 times faster, and CLU_v4, applying all the optimization techniques, is 15.9 times faster. CLU_v3 is also about 4.1

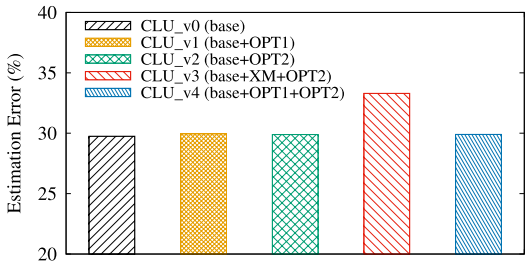


FIGURE 7. Estimation error rate comparison of applying the optimization.

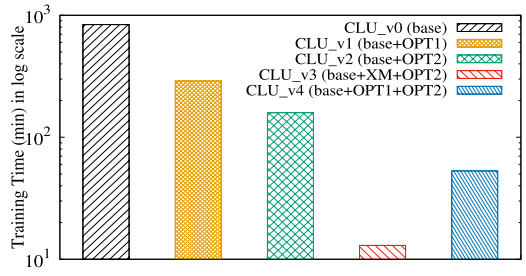


FIGURE 8. Training time comparison of applying the optimization methods.

times faster than CLU_v4 (our proposed optimal model). As mentioned earlier, however, the estimation performance of CLU_v3 is far lower than that of CLU_v4. Therefore, CLU_v4 is the optimal choice; that is, CLU_v4 can save training time by more than one order of magnitude without losing estimation accuracy.

In short, our proposed optimization techniques can reduce training time without affecting accuracy, and the resulting model is more effective than that of the previous study [40].

VIII. CONCLUSION AND FUTURE WORK

In this manuscript we proposed *CLUTCH*, a novel simulation runtime estimation scheme based on an ensemble of clustering, classification, and regression. We also presented two optimization techniques: one determining an optimal permutation of given data pre-processing methods and one finding the optimal k number of clusters in an automated fashion. The optimization techniques significantly reduced the overhead of training the runtime estimation model while preserving the quality of the estimation. To the best of our knowledge, we are the *first* to address the issues of deriving the optimal pre-processing permutation and determining the best k for clustering, concerning the runtime estimation problem.

In our experiments, we demonstrated that the simulation runtime estimation models created for each program had an average error rate of 29.75%. Further, the models were applicable to many simulation programs from diverse CSE areas. The models hold effective in a black box environment in which additional profiling information, such as hardware resources or work queue status, is unavailable.

Importantly, our scheme showed a growth in error improvement of about 14.19% compared to the state-of-the-art work with the best accuracy [18], and a remarkable 2x improvement over the model with the least accuracy [16].

Our findings can help users plan simulations, and help administrators efficiently schedule simulations, by estimating their runtimes and then making users reconsider input values requiring too much runtime. This saves time and resources.

The direction of future research can be summarized as follows. First, we plan to apply the proposed scheme to other domains. Its success in simulation runtime estimation raises the expectation that our proposed scheme could shed light on some other areas of application. It would also be interesting to see whether there is a way of calculating the optimal k other than using sampling and logarithmic regression, since our optimization may not be applicable to some simulation programs. Lastly, developing a job scheduler via the proposed scheme would be of interest.

REFERENCES

[1] C. Witt, M. Bux, W. Gusew, and U. Leser, "Predictive performance modeling for distributed batch processing using black box monitoring and machine learning," *Inf. Syst.*, vol. 82, pp. 33–52, May 2019.

[2] M. Tanash, B. Dunn, D. Andresen, W. Hsu, H. Yang, and A. Okanlawon, "Improving HPC system performance by predicting job resources via supervised machine learning," in *Proc. Pract. Exper. Adv. Res. Comput. Rise Mach.*, Jul. 2019, pp. 1–8.

[3] M. Soysal, M. Berghoff, and A. Streit, "Analysis of job metadata for enhanced wall time prediction," in *Proc. Workshop Job Scheduling Strategies Parallel Process.*, 2018, pp. 1–14.

[4] C. Lee and D. A. Landgrebe, "Analyzing high-dimensional multispectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 31, no. 4, pp. 792–800, Jul. 1993.

[5] W. Li, S. Prasad, J. E. Fowler, and L. M. Bruce, "Locality-preserving dimensionality reduction and classification for hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 4, pp. 1185–1198, Apr. 2012.

[6] D.-W. Kim, K. H. Lee, and D. Lee, "On cluster validity index for estimation of the optimal number of fuzzy clusters," *Pattern Recognit.*, vol. 37, no. 10, pp. 2009–2025, Oct. 2004.

[7] S. Zhou, Z. Xu, and F. Liu, "Method for determining the optimal number of clusters based on agglomerative hierarchical clustering," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 12, pp. 3007–3017, Dec. 2017.

[8] T. Taghavi, M. Lupetini, and Y. Kretchmer, "Compute job memory recommender system using machine learning," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 609–616.

[9] J.-W. Park and E. Kim, "Runtime prediction of parallel applications with workload-aware clustering," *J. Supercomput.*, vol. 73, no. 11, pp. 4635–4651, Nov. 2017.

[10] T.-P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 256–268, Jan. 2020.

[11] J. Bhimani, N. Mi, M. Leeser, and Z. Yang, "FIM: Performance prediction for parallel computation in iterative data processing applications," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 359–366.

[12] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *Proc. IEEE/ACM 11th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2018, pp. 93–102.

[13] F. Nadeem, D. Alghazzawi, A. Mashat, K. Faqeeh, and A. Almalaise, "Using machine learning ensemble methods to predict execution time of e-science workflows in heterogeneous distributed systems," *IEEE Access*, vol. 7, pp. 25138–25149, 2019.

[14] Q. Wang, J. Li, S. Wang, and G. Wu, "A novel two-step job runtime estimation method based on input parameters in HPC system," in *Proc. IEEE 4th Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2019, pp. 311–316.

[15] M. Naghshnejad and M. Singhal, "Adaptive online runtime prediction to improve HPC applications latency in cloud," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 762–769.
[16] S. Pumma, W.-C. Feng, P. Phunchongharn, S. Chapeland, and T. Achalakul, "A runtime estimation framework for ALICE," *Future Gener. Comput. Syst.*, vol. 72, pp. 65–77, Jul. 2017.
[17] A. Tyryshkina, N. Coraor, and A. Nekrutenko, "Predicting runtimes of bioinformatics tools based on historical data: Five years of galaxy usage," *Bioinformatics*, vol. 35, no. 18, pp. 3453–3460, Sep. 2019.
[18] S. Kim, Y.-K. Suh, and J. Kim, "Extes: An execution-time estimation scheme for efficient computational science and engineering simulation via machine learning," *IEEE Access*, vol. 7, pp. 98993–99002, 2019.
[19] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *Proc. IEEE/ACM Perform. Model., Benchmarking Simulation High Perform. Comput. Syst. (PMBS)*, Nov. 2018, pp. 33–44.
[20] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1695–1704.
[21] Z. Wang, K. Fu, and J. Ye, "Learning to estimate the travel time," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 858–866.
[22] K. Fu, F. Meng, J. Ye, and Z. Wang, "CompactETA: A fast inference system for travel time prediction," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 3337–3345.
[23] H. Hong, Y. Lin, X. Yang, Z. Li, K. Fu, Z. Wang, X. Qie, and J. Ye, "HetETA: Heterogeneous information network embedding for estimating time of arrival," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 2444–2454.
[24] S. Heo, S. Cho, Y. Kim, and H. Kim, "Real-time object detection system with multi-path neural networks," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 174–187.
[25] S. Reder, F. Kempf, H. Bucher, J. Becker, P. Alefragis, N. Voros, S. Skalistis, S. Derrien, I. Puaut, O. Oey, T. Stripf, C. Ferdinand, C. David, P. Ulbig, D. Mueller, and U. Durak, "Worst-case execution-time-aware parallelization of model-based avionics applications," *J. Aerosp. Inf. Syst.*, vol. 16, no. 11, pp. 521–533, Nov. 2019.
[26] S. Currim, R. T. Snodgrass, Y.-K. Suh, R. Zhang, M. W. Johnson, and C. Yi, "DBMS metrology: Measuring Query time," in *Proc. Int. Conf. Manage. Data*, 2013, pp. 421–432.
[27] S. Currim, R. T. Snodgrass, Y.-K. Suh, and R. Zhang, "Dbms metrology: Measuring Query time," *ACM Trans. Database Syst.*, vol. 42, no. 1, pp. 1–42, 2016.
[28] Z. Chu, J. Yu, and A. Hamdulla, "A novel deep learning method for Query task execution time prediction in graph database," *Future Gener. Comput. Syst.*, vol. 112, pp. 534–548, Nov. 2020.
[29] D. Ardagna, E. Barbierato, E. Gianniti, M. Gribaudo, T. Pinto, A. da Silva, and J. Almeida, "Predicting the Performance of big data applications on the cloud," *J. Supercomput.*, May 2020, doi: 10.1007/s11227-020-03307-w.
[30] A. D. Popescu, A. Balmin, V. Ercegovac, and A. Ailamaki, "PREDIcT: Towards predicting the runtime of large scale iterative analytics," *Proc. VLDB Endowment*, vol. 6, no. 14, pp. 1678–1689, Sep. 2013.
[31] D. K. V., D. R., and B. A., "Artificial bee colony algorithm for grid scheduling," *J. Converg. Inf. Technol.*, vol. 6, no. 7, pp. 328–339, Jul. 2011.
[32] M. Pal, "Random forest classifier for remote sensing classification," *Int. J. Remote Sens.*, vol. 26, no. 1, pp. 217–222, Jan. 2005.
[33] A. Liaw and M. Wiener. (2017). *Package: Randomforest*. [Online]. Available: https://cran.r-project.org/web/packages/random Forest/randomForest.pdf
[34] (2020). *Kmeans R Package*. [Online]. Available: https://www.rdocumentation.org/packages/stats/versions/3.6.1/topics/kmeans,
[35] Q. Zhang and I. Couloigner, "A new and efficient K-medoid algorithm for spatial clustering," in *Int. Conf. Comput. Sci. Aopl.*, 2005, pp. 181–189.
[36] R Core Team. (2016). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. [Online]. Available: https://www.R-project.org/
[37] Y.-K. Suh, H. Ryu, H. Kim, and K. W. Cho, "EDISON: A Web-based HPC simulation execution framework for large-scale scientific computing software," in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGrid)*, May 2016, pp. 608–612.
[38] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *Int. J. Forecasting*, vol. 22, no. 4, pp. 679–688, Oct. 2006.
[39] T. Tsutsui and T. Matsuzawa, "Virtual metrology model robustness against chamber condition variation using deep learning," *IEEE Trans. Semicond. Manuf.*, vol. 32, no. 4, pp. 428–433, Nov. 2019.
[40] D. Pelleg and A. W. Moore, "X-means: Extending K-means with Efficient Estimation of the Number of Clusters," in *Proc. 17th Int. Conf. Mach. Learn.*, 2000, pp. 727–734.

YOUNG-KYOON SUH (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, The University of Arizona, in 2015. From 2005 to 2017, he was a Senior Researcher with the Korea Institute of Science and Technology Information. Since 2017, he has been an Assistant Professor with the School of Computer Science and Engineering, Kyungpook National University, Daegu, Republic of Korea. His research interests include databases, big data systems, machine learning, high-performance computing, and performance evaluation. He is a member of ACM. He was a recipient of the Best Poster Award of IEEE/ACM CCGrid 2016 and the Best Paper Award of EDB 2016.

SEOUNGHYEON KIM received the B.S. and M.S. degrees from the School of Computer Science and Engineering, Kyungpook National University, Daegu, Republic of Korea, in 2018 and 2020, respectively. He is currently an Engineer with Samsung Research. His research interests include data mining and machine learning.

JEEYOUNG KIM received the Ph.D. degree in computer and information science and engineering from the University of Florida, Gainesville, FL, USA, in 2013. From 2013 to 2018, she was a Senior Engineer with the Mobile Division, Samsung Electronics. From 2018 to 2019, she held the role of a Research Faculty with the Center of Self-Organizing Software, Kyungpook National University, Daegu, Republic of Korea. Since 2019, she has been a Teaching Faculty with the School of Computer Science and Engineering, Kyungpook National University. Her research interests include the IoT, real-life human generated data, machine learning, and big data systems.