937
Towards Optimal Simulation Jobs Scheduling Using Machine Learning Techniques for Predicting Simulation Runtime

Sulaiman Al-Gannas, Razen Al-Harbi, Ghaidaa Al-Kuhaili, Ali Al-Turki

Saudi Aramco, Dhahran, KSA

EXTENDED ABSTRACT

Introduction

In order to improve the efficiency of a High-Performance Computing (HPC) simulation environment, it is necessary to provide accurate estimation of both compute resources requirements and runtime (execution time). Due to lack of alternatives, the runtime assignment is a human input, which is prone to error and bias; hence, it is considered unreliable. Accordingly, the underlying compute resource manager, (i.e. job scheduler) cannot utilize its sophisticated scheduling mechanisms in order to effectively minimize wait time and maximize the HPC resource utilization. Therefore, an accurate prediction of simulation job's runtime is considered an essential prerequisite for efficient scheduling. This abstract detail the development of a comprehensive Machine Learning (ML) model to accurately predict the runtime of simulation jobs.

Capitalizing on Iterative and Automated Machine Learning (AutoML) techniques, several ML models were developed and evaluated to accurately predict the simulation job runtime. The development and evaluation criterion followed a consecutive order. First, a partitioning feature is selected and used for partitioning the data into several data samples. On each partitioned data sample, a set of ML models are created using several algorithms (e.g. Decision Tree, Random Forest, deep-learning) that fits the runtime prediction problem and has the potential of yielding high prediction accuracies. The ML model with the highest accuracy is selected for predicting that data sample. As a result, a set of ML models are created instead of a single model. At prediction stage, the partitioning value will serve as the selector for the proper ML model. This comprehensive ML pipeline is built in a fully automated manner.

Predicting the job's total runtime is considered an extremely challenging problem for highly parametrized datasets. It has been found that each of the segregated models is overfitting the assigned data sample. This was mitigated by using cost complexity pruning with 10-fold cross validation which resulted in simplifying the model complexity and

increasing the model prediction quality. To achieve even more reliable prediction models, the simulation job runtime is transferred from continues values (i.e. seconds, minutes, hours) into meaningful intervals. This helped in having a supervised classification model which can be auto-tuned effectively in future.

The new ML models were successfully trained using more than 200K simulation jobs and managed to predict the simulation runtime with an accuracy of up to 90%.

Unlike existing ML techniques which rely on a single model classification, the followed approach creates an ensemble of ML models and utilizes the one that yields the most accurate one during prediction. Consequently, this will help the underlying HPC resource manager to minimize the job waiting time and maximize the HPC resource utilization. Also, it will help the users in building a clear understanding on the job turnaround time and plan their activities accordingly.

Job scheduling in a multiprocessing environment is a dynamic process which allows managing, scheduling, and monitoring parallel running jobs to maximize the resource allocation. Once a job is submitted, it is put into a job queue, which consists of all jobs waiting for execution. Based on the amount of resources the job is requiring, the scheduler will look for availability in free resources. If the amount of resources required are free, the job will start executing and move from the ready state to the running state. Otherwise, the job will wait until the needed resources are all free. Waiting time will vary based on the execution time of already submitted jobs and the amount of resources a cluster has. As the job moves through the simulation environment, it is always in one of the job states as shown in **Fig. 1** (Ann et al. 2013).

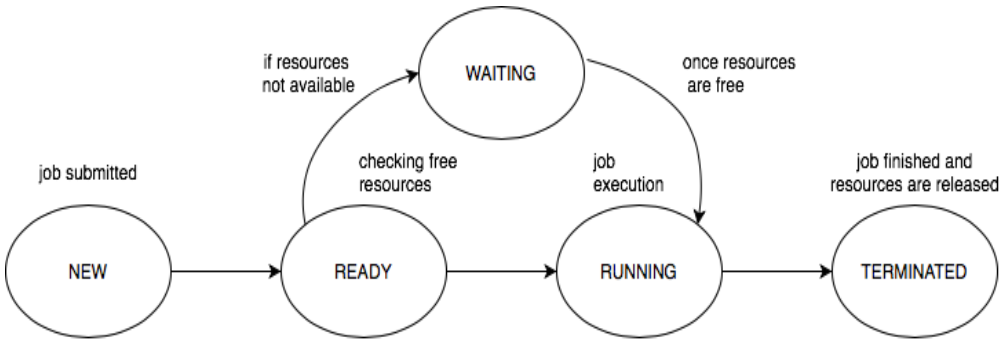

Fig. 1 **–** A typical job changes state as it moves through the simulation environment from NEW to TERMINATED

As jobs are waiting for their turn to start execution, new jobs are being submitted to the simulation environment. Some of the jobs in the running state are completed and hence terminated. On the other hand, some running jobs requires more time than what has been scheduled for them. This interruption is caused by the time input provided by the user. The user spontaneously assumes the amount of time the job will take to finish execution. However, this assumption is not always accurate and is prone to error. In case of Hard real-time scheduling, this uncertainty will cause catastrophic consequences, where deadlines are critical and should be met in all working scenarios or otherwise the job will be killed and

terminated. In contrast, Soft real-time scheduling, where the harm is low and missing the deadlines is less critical and has minimal effect on the scheduling performance.

Jobs in the waiting queue are prioritized according to a number of factors. The schedulers will then start executing the jobs one by one stepping through the priority list until it reaches a job which cannot start due to the unavailability of free resources. Backfilling algorithms allow small jobs from the back of the queue to execute before larger jobs that are stepping at the beginning of the queue. This will occur only if the amount of resources required by the small jobs are available and will not delay the run of already scheduled larger jobs. By doing so, jobs will run out of order allowing the execution of larger number of parallel jobs which will increase the resource utilization and hence improve the performance.

Every machine learning service needs to solve fundamental problems of deciding which machine learning algorithm to use on a given dataset, whether and how to preprocess its features, and how to set all hyperparameters. Implementing such ML tasks require a deep or relatively intermediate level of experience in the field. As the complexity of these tasks is often beyond non-Machine Learning -experts, the rapid growth of machine learning applications has created a demand for off-the-shelf machine learning methods that can be used easily and without expert knowledge. The resulted demand i.e. research area is called Automated Machine Learning. Which will provide non-Machine Learning experts methods and processes to accelerate the research area.

Machine-Leaning Model Development

To design a robust machine learning model, we chose scikit-learn. One of the best known and widely used machine learning libraries (Fabian et al.2011). Fig. 2 depicts the system architecture of the configuration space used for building, training, and testing the machine learning model.

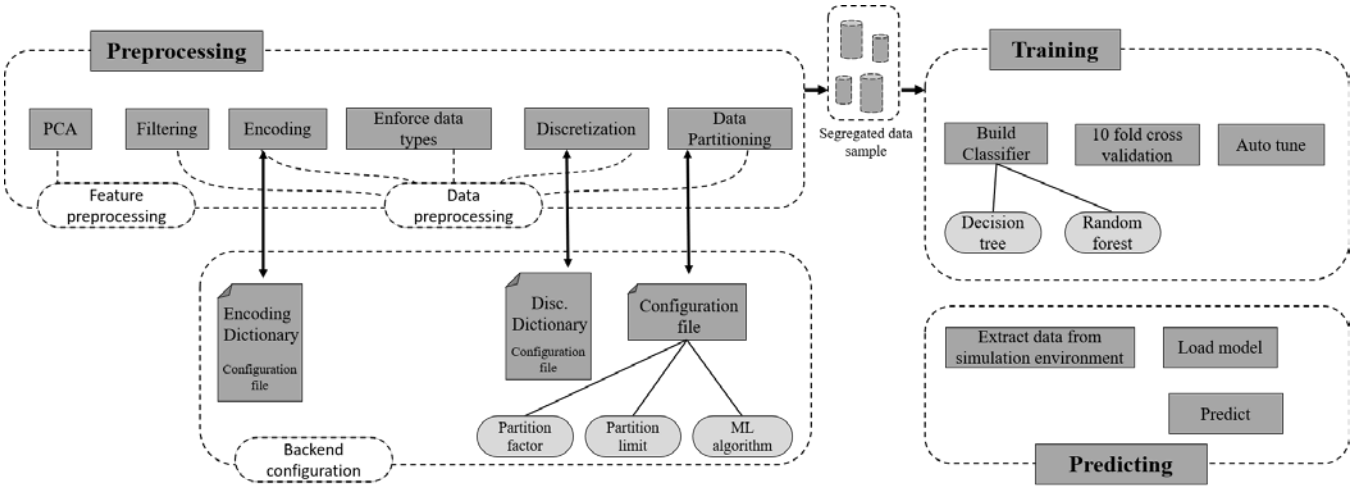

Fig. 2 – The structured configuration space for the designed machine learning model

Three main functionalities are performed in the proposed system architecture. We can think about them as three different pipelines. Each pipeline will significantly contribute in building the ML model. The first pipeline is responsible for performing the preprocessing phase. This phase provides a reliable starting framework for cleaning the data and maintaining a healthy data environment to be used in the proceeding steps of building the model. In order to reduce the high dimensional spectral of features we have in the raw data, a principle component analysis (PCA) was performed. Features with high correlation with the class feature are selected. This ended up reducing the number of features used in building the model from 74 to 29 features.

To ensure including only the successfully completed simulation runs, filtering techniques were implemented to keep only sufficient samples that will add a value to our analysis. As the stipulated encoding methodologies i.e. One- Hot encoding, Dummy Variable encoding, etc. were not supportive in our case, categorical features were encoded manually. Based on the tangible effect of each category, it will be given a specific magnitude. Such that, the higher the effect to the simulation run time, the greater the magnitude will be. The key values assigned to each categorical feature are stored in a configuration file that will be fetched during data cleaning phase. Since the targeted feature (class) is a continues numerical value i.e. Total Runtime, this made it difficult and computationally expensive to perform prediction. A discretization technique was implemented to convert the time data points into meaningful intervals. By doing so, the class column is converted from continues numerical data points into categorical numerical data points, and hence converting the prediction problem from regression into a supervised classification problem. The intervals boundaries are defined in a separate configuration file that will be called during the data cleaning phase.

The preprocessing phase will end up with partitioning the data sample based on the partitioning factor defined by the user. This factor will be instantiated by the user in a configuration file, and it will take the name of the column (feature) that the user would like to

use in portioning the data. Since the partitioning factor is configured in a configuration file, changing it will not cause changing the data cleaning code. The partitioning process will depend on the partitioning limit factor. The user will have to define a limit for the minimal number of records (rows) a sample will take. If a sample contains a number of records less than what has been specified, a default sample will be used in this case. The default sample is prepared at an early stage where it will contain a sufficient number of data records. **Fig. 3** simplifies the partitioning concept.

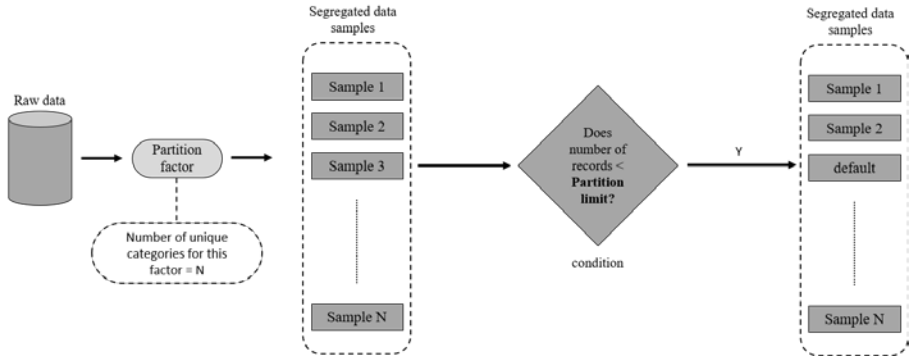

Fig. 3 – Each sample will pass through the validation step and if it satisfies the condition, the default sample will be used instead

The segregated data samples will be fed as an input to the second pipeline, which is responsible for performing the training phase. The training phase will testify building, training, testing, and validating the ML models. For each data sample, three ML models will be built. One using the decision tree algorithm, second using the random forest algorithm, and the third using deep-learning. The number of models and the type of algorithms used in building the models can be monitored in the same configuration file used in defining the partitioning factor. Unlike Auto-ML, the user will have the ability to specifically define the algorithms that will best help in achieving the prediction objective. At this point, each sample will have three different models. A 10-Fold cross validation is implemented on each model to evaluate its performance. An auto-tuning process is implemented to select the best model for each sample. Each data sample will end up having one best model that will be used further in the prediction process. **Fig. 4** summarizes the steps occurs in the training phase.

Deploying the final models will finalize the training phase and make our ML models ready to execute the third pipeline, which is responsible for performing the prediction phase. When a new record is received, the system will check for the partitioning factor and load the appropriate corresponding model to be used in predicting the simulation run time.

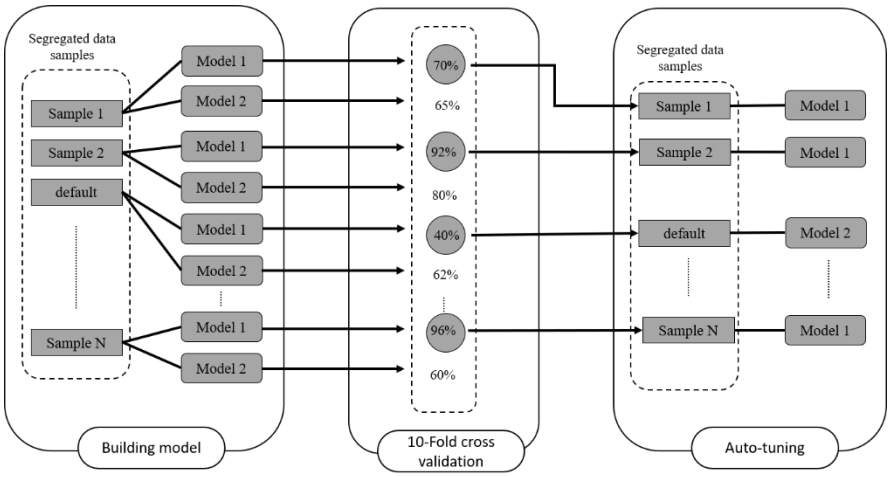

Fig. 4 – The phase starts with building the N models, and then perform 10-Fold cross validation to calculate the accuracy of each model, and finally implements auto-tuning by selecting the best model

Results

Capitalizing on Iterative and Automated Machine Learning (AutoML) techniques, several ML models were developed and evaluated to accurately predict the simulation job runtime. Deploying the final models will finalize the training phase and make our ML models ready to execute the prediction phase. When a new record is received, the system will check for the partitioning factor and load the appropriate corresponding model to be used in predicting the simulation run time. The accuracy of the predicted runtime is 90% and improving with more data being collected. **Table 1** displays a sample of models showing predicted versus actual runtime results.

Table 1 - The predicted versus actual runtime of sample simulation models running HPC cluster

models	predicted class	predicted runtime (Hours)	actual runtime (Hours)
case1	Class K	60-80	73
case2	Class C	5-7.5	7
case3	Class B	2.5-5	5
case4	Class I	40-50	43
case5	Class D	7.5-10	8
case6	Class F	15-20	17
case7	Class B	2.5-5	4
case8	Class B	2.5-5	3
case9	Class B	2.5-5	4
case10	Class K	60-80	61

case11	Class B	2.5-5	2
case12	Class B	2.5-5	4

Conclusion

Predicting the job's total runtime is considered an extremely challenging problem for highly parameterized datasets. It has been found that each of the segregated models is overfitting the assigned data sample. This was mitigated by using cost complexity pruning with 10-fold cross validation which resulted in simplifying the model complexity and increasing the model prediction quality. To achieve even more reliable prediction models, the simulation job runtime is transferred from continuous values (i.e. seconds, minutes, hours) into meaningful intervals. This helped in having a supervised classification model which can be auto-tuned effectively in the future. The new ML models were successfully trained using more than 200K simulation jobs and managed to predict the simulation runtime with an accuracy of up to 90%.