A Runtime Estimation Framework for ALICE

Sarunya Pumma[a*], Wu-chun Feng[a], Phond Phunchongharn[b], Sylvain Chapeland[c], and Tiranee Achalakul[b]

[a]*Department of Computer Science, Virginia Tech, USA*
[b]*Department of Computer Engineering, King Mongkut's University of Technology Thonburi, Thailand*
[c]*Department of Physics, European Organization for Nuclear Research (CERN), Switzerland*

Abstract

The European Organization for Nuclear Research (CERN) is the largest research organization for particle physics. ALICE, short for *A* *L*arge *I*on *C*ollider *E*xperiment, serves as one of the main detectors at CERN and produces approximately 15 *petabytes* of data each year. The computing associated with an ALICE experiment consists of both online and offline processing. An online cluster retrieves data while an offline cluster farm performs a broad range of data analysis. Online processing occurs as collision events are streamed from the detector to the online cluster. This process compresses and calibrates the data before storing it in a data storage system for subsequent offline processing, e.g., event reconstruction. Due to the large volume of stored data to process, offline processing seeks to minimize execution time and data-staging time of the applications via a two-tier offline cluster — the Event Processing Node (EPN) as the first tier and the World LHC Grid Computing (WLGC) as the second tier. This two-tier cluster requires a smart job scheduler to efficiently manage the running of the application. Thus, we propose a runtime estimation method for this offline processing in the ALICE environment.

Our approach exploits application profiles to predict the runtime of a high-performance computing (HPC) application without the need for any additional metadata. To evaluate our proposed framework, we performed our experiment

*Corresponding author: Phond Phunchongharn

on the actual ALICE applications. In addition, we also test the efficacy of our runtime estimation method to predict the run times of the HPC applications on the Amazon EC2 cloud. The results show that our approach generally delivers accurate predictions, i.e., low error percentages.

Keywords: `Runtime Estimation`, `ALICE Experiment`, `Berkeley Dwarfs`, `Offline Scheduling`, `Scheduler`, `Workload Characterization`

1. Introduction

Currently, the European Organization for Nuclear Research (CERN) is the world's largest research organization for particle physics. Its most recent particle accelerator is the Large Hadron Collider (LHC), which serves to boost the energy of particles to be close to the speed of light. Inside the LHC, two proton beams travel in opposite directions in the separated pipes until they are allowed to cross each other at the detectors, where the collisions between particles occur. An enormous number of collision events, in the order of 600-million collisions per second, are detected and recorded by the detectors located along the LHC ring.

ALICE, A Large Ion Collider Experiment, is a heavy-ion detector for studying the physics of strongly interacting matter at the CERN LHC [1]. In particular, it targets the analysis of the properties of Quark-Gluon Plasma, using proton-proton, nucleus-nucleus, and proton-nucleus collisions at high energies. In 2018, the ALICE detectors will be upgraded [2, 3], and the associated amount of data that will be produced from the detectors will increase by an *additional two orders of magnitude*, resulting in a data throughput of approximately 1 TB per second. In order to keep up with this data deluge, a more powerful and intelligent computing system must be designed and realized.

This new computing system includes the design, implementation, and optimization of both online and offline processing capabilities, as outlined by the data flow in Figure 1. The detectors and online cluster farm normally operate only 4-8 months per year; the rest of the time is dedicated to offline processing.

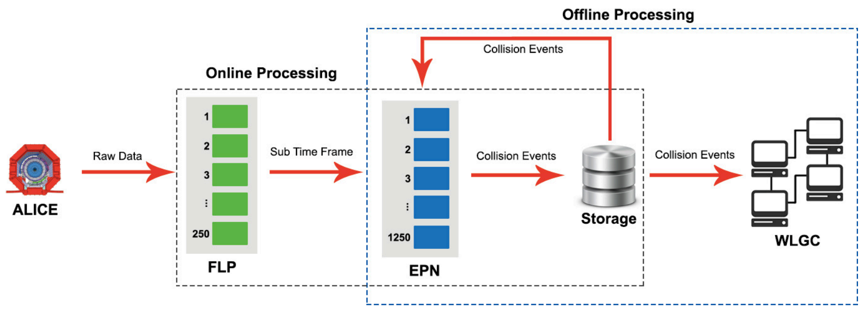

Figure 1: ALICE's Online and Offline Processing Data Flow

In ALICE, the online process receives collision events from the detectors and stores them for further processing. Because the amount of incoming data will increase substantially in the next phase of LHC in 2018 (referred to as *Run3*), this process will then have to compress and control the data rate so as to not exceed the capability of the storage system, which provides a data rate of 200 GB/s at peak and 50 GB/s on average.

Based on the data flow from Figure 1, our online data acquisition consists of two compute clusters — First-Level Processors (FLP) and Event Processing Nodes (EPN). The FLP cluster receives the collision events, which are grouped in a timeframe spanning 0.1 seconds. The resulting data rate is then 100,000 collision events per second or 10 timeframes per second. Due to the limited network bandwidth, FLPs reduce the data by approximately a factor of *five* and stream it to the EPN cluster. The EPN cluster then aggregates the streamed sub-time frames into full-time frames, reduces the data size by an additional factor of *four*, and then calibrates the data before storing it in the storage system.

The EPN cluster also processes offline tasks, which include event reconstruction, event calibration, event simulation, and data analysis. The offline processes run on EPN when it is unoccupied by the online processes. Since there are a large number of applications running on the EPN cluster, an efficient scheduler is required to manage job executions. The scheduler has to be able to assign the jobs to efficiently run on machines in the EPN cluster. When EPN is not available, the offline processes are assigned to the Worldwide LHC Computing Grid (WLCG) as a second-tier (alternative) cluster. The preference, however, is to run offline jobs on the EPN cluster rather than on WLGC.

In this work, we focus on the scheduling of offline jobs on the EPN cluster as WLCG already has its own job scheduler, namely gLite [4]. Our offline scheduler seeks to run the offline jobs on the EPN cluster as efficiently as possible, as running them on WLCG is more expensive. To implement an efficient scheduler, we need to predict the runtimes of applications. However, predicting the runtimes of the applications in a computer system is a daunting challenge. To ease this challenge, some computer systems request the expected runtime from the user. Although this method is easy, it is inaccurate and inefficient due to overestimation [5]. In addition, some existing methods for runtime estimation assume that the same user runs the same application in the system [6]. Therefore, a user name and a project are used as a key. If the application submitted to the system has the same key, the runtime is calculated from the actual runtime of the previous run. However, the key used in this method is not as informative as it does not contain any runtime behavior of the application.

We propose a methodology to efficiently estimate the runtime of "black-box" applications in the ALICE experimental environment. The "black-box" processes are the applications in which their source codebases are *not* available. In contrast to other approaches, we extract important application characteristics, which capture execution behavior, to predict the runtime for an application. When the application is submitted to the EPN cluster, these characteristics are sampled for a short period by using workload characterization tools, namely MICA, short for *Microarchitecture-Independent Characterization of Applications*l [7], and the `perf` tool from the Linux kernel [8], in order to create a workload profile. Our methodology then creates a model for workload classification, followed by a model for runtime prediction. The input to both of these modeling steps is a set of performance metrics collected from the MICA and `perf` tools. The workload classification model categorizes the application into a certain class based on its characteristics. The characteristics of the applications in each class are classified with respect to the Berkeley Dwarfs taxonomy [9], where each class of applications has a separate runtime prediction model on a specific type of machine. Our prediction model uses the *Artificial Bee Colony (ABC)* [10] optimization in concert with linear regression.

In turn, the runtime of the workload can then be predicted without any additional information about the applications. Furthermore, once our runtime-prediction models are constructed, they are reusable. Thus, the runtimes of the applications in the EPN cluster can be estimated immediately and automatically. In addition, the estimation models can be re-calibrated as additional data sets become available.

The rest of the paper is organized as follows. In Section 2, we present related work. Section 3 describes our proposed runtime estimation framework. In Section 4, we present our experiments and experimental results. Section 5 discusses some limitations of our runtime prediction framework. Section 6 concludes the paper.

2. Related Work

The runtime of an application is an important attribute in many scheduling schemes, e.g., backfilling [11]. Backfilling requires an application's runtime to insert short jobs in the available slots without delaying higher-priority jobs. Consequently, the accuracy of runtimes in backfilling is critical in realizing an efficient scheduled system. Oftentimes, the runtimes of applications in a system are provided by users. However, the user typically overestimates the runtimes. [6] has shown that approximately 50% of 275,000 jobs in the system used only half of the user-estimated runtime to finish their jobs. This results in degrading the overall efficiency of the system [5]. Therefore, a large body of work addresses the user runtime overestimation problem [12, 13, 14, 6].

For user characteristic based approaches [12, 13, 14, 6], Tsafrir et al. [12] improved a scheduler's performance by using the user's estimated runtime as an upper bound and predicting a runtime by averaging the runtimes of the last two jobs of the same user. Minh and Wolters [13] estimated a runtime of a job based on K-nearest neighbors by aiming to reduce a number of jobs that were under-estimated while maintaining a good runtime prediction accuracy. Gaussier et al. [14] replaced a user runtime prediction of the EASY backfilling scheduler with the *l2*-regularized polynomial model and gained 28% performance improvement. Another user-dependent runtime prediction method was proposed by Tang et al. [6]. The authors proposed a methodology to adjust the user-provided runtime in order to improve the performance of a supercomputer system. Their adjustment scheme searched for a similar application based on keys (i.e., a user name of the user who submitted the job and the project name) from the historical data and calculated an adjustment factor (R) by averaging the R-values of similar applications. This approach is still based on the assumption that the same user will submit the same project with the same input and with the same level of over-estimation. However, this assumption would not be practical in some cases. Instead, the runtime of an application should depend on the characteristics of the applications rather than the users.

For application characteristic based approaches, a large number of estimators exploit historical data to infer runtimes [15, 16, 17, 18]. Krishnaswamy et al. [15] estimated the computation times for data-intensive applications by using the mean runtimes of similar applications. The similarity between applications could then be determined by rough sets theory, which uses the historical data to find the subset of attributes that strongly relate to the runtimes. The output of rough sets is a similarity template. Smith et al. [16] also implemented their runtime prediction framework based on similarity templates, which could be determined by greedy and genetic algorithms. The runtime of the application could be derived in two ways: (1) using the mean of the runtimes or (2) using the linear equation to calculate the runtime. However, the similarity template could only work well in the scenarios in which similar applications are repeatedly executed in the system.

Xia et al. [17] predicted the runtime from the historical information stored in the form of cases. The cases were defined by using the TA3 algorithm, a case-based reasoning approach, which determines the runtime using the average value of the runtimes. The drawback, however, is that the number of cases is not predefined and can grow without bound, which in turn, could significantly negatively effect the performance of the system. Thus, the policy to control the number of cases must be well defined for this approach to be effective. Zhang et al. [18] proposed a resource-oriented approach that predicts the runtime of the workloads in a grid environment by extracting information about the resources from the Grid Information System (GIS), namely CPU load. The CPU load is then fed into a time-series model to predict the estimated runtime. However, this method requires an accurate value for the CPU load for the application. The approaches in [15, 16, 17, 18] require some historical computational data or attributes of the applications for similarity identification. However, certain sets of attributes used in workload classification or clustering are not explicitly defined. Consequently, common attribute sets need to be defined to improve the performance of workload similarity identification.

Since the relationship between application characteristics and runtime is not explicit, machine learning techniques have been widely used in performance and runtime prediction frameworks [19, 20, 21, 22, 23, 24, 25]. Kadirvel and Fortes [19] proposed a grey-box machine learning based approach to estimate performance of Map-Reduce platforms. This work explored multiple machine learning techniques, for example, Gaussian Process Regression and Multilayer Perceptron, and showed that they outperformed typical regression approaches, such as simple linear regression, in an aspect of accuracy. Kousiouris et al. [20] predicted non-deterministic black-box user behaviors using the neural network in the Software-Platform-Infrastructure (SPI) cloud to efficiently provision low-level resources to guarantee the quality of service (QoS). Prodan and Nae [21] predicted load of Massively Multiplayer Online Games based on historical data series. The neural network with a sliding window method (where the training input was the set of data points within the window) was presented to predict the sudden surge of resource usages in the cloud computing platforms to proactively provision resources to prevent a severe delay in response time [22]. Li et al. [23]

proposed a function-specific runtime prediction model using the artificial neural network. Although the approach could provide a promising accuracy, it was not practical for programs with a large number functions to have one model for one function since the training time could be enormous as well as the prediction time. In [24] and [25], the Predicting Query Runtime Regression (PQR2) method, which is a binary tree-based approach, was used to generate a runtime estimation model. One of the drawbacks of the model is that it is application-specific.

Based on the control factors for generating the prediction models, we can divided the previous works into three main groups which are Cluster specific, Machine type specific, and Application specific. For the Cluster specific group [6, 12, 13, 14, 15, 16, 17, 26], the prediction models were generated by using information gathered from a specific cluster which could be either homogeneous or heterogeneous. Therefore, the cluster environment must be controlled. The predication models assumed that jobs from the same users are similar. However, the actual runtimes depend on both the application characteristics and machine specifications. Consequently, the mean absolute error percentages (MEAP) of Cluster specific group ranged from 15% to 45%. To improve the accuracy of runtime prediction, the prediction models using machine learning techniques for specific machines (called Machine type specific group) were proposed in [19, 20, 25]. Since the machine specification was controlled and machine learning techniques could adaptively learn and detect the patterns of jobs and machine behaviors, the MEAP of Machine type specific could be improved to the range between 10% and 20%. Finally, [21, 22, 23, 24] proposed the runtime prediction models specifically to applications, called Application specific group. Although these approaches utilized particular application characteristic data to predict runtimes, there were no significant improvement of the MEAP (range between 5% and 30%). Moreover, runtime prediction models in this group were too limited. The specific predication models must be constructed for each application. In practice, there is a broad range of applications executed in a machine. The approaches in this group would therefore not be suited for such the systems.

Since the ALICE system consists of various types of physics applications to run on specific types of machines, we focus on a generic runtime estimation model that can predict a runtime for any types of applications that have similar characteristics on a specific machine. Although machine learning techniques in [19, 20, 21, 22, 23, 24, 25] can provide a promising accuracy in runtime estimation, these techniques are not suitable for a dynamic environment. If the characteristics of applications are constantly changing so the learning models must be retrained and the prediction models must be regenerated. Consequently, we propose a meta-heuristic optimization algorithm together with classification and regression technique to estimate runtime accurately and robustly in dynamic environments.

Unlike other work, we use Aritificial Bee Colony (ABC) [10], a meta-heuristic artificial intelligence approach, collaborating with the linear regression technique to construct a runtime estimation model based on an informative set of attributes. The ABC algorithm has been chosen because, based on several research papers [27, 28, 29], it can produce a better optimal solution than other approches such as Particle Swarm Optimization (PSO), Evolutionary Algorithm (EA), and Genetic Algorithm (GA). For the informative set of attributes, we obtain them from MICA (Microarchitecture-Independent Characterization of Applications) [7], an analysis tool for capturing the profile of workloads on computer systems, and perf [8] from the Linux kernel. These attributes can capture the execution behavior of the applications. Consequently, our proposed framework can adaptively estimate the runtime in dynamic environments. With the attributes from MICA and perf, we classify workloads based on the taxonomy of the Berkeley Dwarfs. Relative to the Berkeley Dwarfs, the similarity in computation behavior and data flow can be used to define membership in the class [30]. Currently, there are 13 dwarfs [31]. Of the 13 dwarfs, we realized only seven classes of the dwarfs and eliminated the remaining classes as they would produce redundant characteristics. These seven (7) dwarfs are able to represent classes of most applications in the ALICE system.

3. Runtime Estimation Framework

We propose an efficient runtime estimation framework for offline jobs, especially in the EPN cluster of the ALICE system. Our proposed framework contains three main steps as illustrated in Figure 2. In Step 1 Profile Sampling, a "black-box" application from the ALICE experiment is submitted to our system and runs for a small period of time. MICA and pert tools are deployed to create a sample profile of the application. To elaborate, application behavior is profiled based on a set of parameters, such as percentage of multiply instructions, branch predictability, and probability of a local and global load and store (The full list of parameters is shown in Table 1). These parameters' values are captured for each application during runtime using MICA and perf tools. In Step 2 Workload Classification, the captured data is fed into a decision tree in order to classify the application into one of the Berkeley Dwarf classes with the most similar runtime behavior. Note that using only seven out of thirteen Dwarf classes is sufficient to represent applications in the ALICE system. Table 1 shows the abbreviated notation for the seven classes used in this paper. In Step 3 Runtime Estimation, the application runtime is predicted using the regression model of the dwarf class that the application belong to. The runtime models are described in a set of mathematical equations. The following subsections provide detailed descriptions of our methodology and validation.

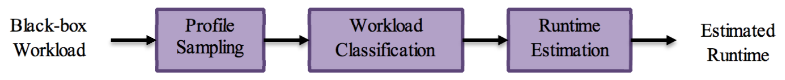

Figure 2: The overall methodology of the runtime estimation framework

3.1. Profile Sampling

The "Black Box" workload is profiled using twelve parameters as listed in Table 2. The top eight parameters are microarchitecture-independent metrics collected using MICA, while the bottom four are system parameters collected using `perf`. These 12 quantitative metrics truly represents runtime characteristics of applications. Moreover, our profile sampling method only relies on the

Table 1: Dwarf list

Dwarf Name	Notation
Dense linear algebra	*dense*
Sparse linear algebra	*sparse*
Spectral methods	*spectral*
N-body methods	*nbody*
Structured grids	*sgrid*
MapReduce	*mapred*
Graph traversal	*grapht*

parameter values of the current run. No assumption is made on users' previous runs. Thus, no background knowledge on usage pattern is needed.

Algorithm 1: Profile Sampling

```
1 run application under MICA environment while application is running
    do do
2     for every 1 million instruction do
3         MICA collects architecture-independent parameter values
4     end
5     for every 2 second do
6         run perf to collect architecture-dependent parameter values
7     end
8 end
```

The profile sampling step has to be performed separately for the Training and Testing phases in the proposed runtime estimation framework.

- In the training phase, we collect the profile sample of 20 benchmarks offline. The benchmarks come from three standard suites, which are Rodinia [32], NPB [33], and TORCH [34]. These benchmarks are good representations of most scientific applications. Algorithm 1 shows the steps

Table 2: List of metrics

Tool	Metric	Notation
MICA	1. Probability of a register dependence distance $\leq$ 16	$P_{RegDist \leq 16}$
	2. Branch predictability of per-address, global history table (PAg) prediction- by-partial-matching (PPM) predictor	$B_{predict}$
	3. Percentage of multiply instructions	Pct_{mult}
	4. Data stream working-set size at 32-byte block level	WSS
	5. Probability of a local load stride = 0	$P_{LRStride=0}$
	6. Probability of a global load stride $\leq$ 8	$P_{GRStride \leq 8}$
	7. Probability of a local store stride $\leq$ 8	$P_{LWStride \leq 8}$
	8. Probability of a local store stride $\leq$ 4,096	$P_{LWStride \leq 4096}$
Perf	9. CPU clock	CLK_{CPU}
	10. Task clock	CLK_{Task}
	11. Page faults	PF
	12. Context switches	CS

of profile sampling. Each benchmark is executed multiple times until completion with various settings of input parameters, input sizes, and runtime. During each execution, the top eight parameters (architecture-independent) from Table 2 are collected every 1 million instructions via MICA. The bottom four parameters (architecture-dependent) are collected every 2 seconds using `perf`. These values are the profile of the benchmarks, which will be used to construct the classification model in the next step.

- In the testing phase, the profiles of the workloads from the ALICE system are created in real time using a method presented in Algorithm 1. The test applications only run for a small window of time. The profile is fed to the next step for classification.

3.2. Workload Classification Model

In this step, the pre-constructed classification model is used to categorize an unknown-profile workload from the previous step into one of the dwarf classes listed in Table 1. The classification result is further utilized in the runtime prediction step. Section 3.2.1 and 3.2.2 explain the details of this step for the Training (model construction) and the Testing (model testing)) phases in our proposed framework respectively.

3.2.1. Construction of the Workload Classification Model

In order to construct a classification model, we prepared a set of feature vectors to be used as a training data set. We used the 255 different workload profiles collected in the previous step and label them into Dwarf classes using information provided in the related works [32, 34, 35]. Twenty applications from three benchmark suites were mapped into Berkeley Dwarf classes as shown in Table 3. Note that only the architecture-independent (AI) metrics were used in the feature vectors because the characteristics of the algorithm do not depend on the system architecture. Thus, each feature vector consists of 8 floating point numbers (0-1), representing AI metrics, and a class label.

Table 3: Mapping between benchmarks and dwarfs

Dwarf	Kernel/Application		
	Rodinia	NPB	TORCH
dense	kmeans lud nn	lu(A,B,C,S)	dense
sparse	-	cg(A,B,C,S)	sparse
hline *spectral*	-	-	spectral
nbody	-	-	nbody2d
sgrid	heartwall hotspot lavaMD leukocyte particle	sp(A,B,C,S)	-
mapred	-	ep(A,B,C,S)	monteCarlo
grapht	-	-	integerSort quickSort radixSort

We have selected the C4.5 decision tree algorithm as our model construction method. C4.5 has a low overhead, is easy to interpret, and is widely used in real applications [36, 37]. To construct a decision tree, the Weka data mining analysis tool is used. During this training phase, 255 feature vectors were fed as inputs into C4.5. The tree was formed and self-adjusted until the training phase was finished. The output of the C4.5 algorithm is a decision tree that can be linearized into a set of decision rules. These sets of rules can be used to classify applications into Dwarf classes. There are 7 rules generated for 7 Dwarfs. Each rule is a Boolean expression of MICA's metrics. The application belongs to a Dwarf class if a set of conditions on MICA metric values fit the rule of that class.

To measure the accuracy of the decision tree, we apply a stratified 10-fold cross-validation to the model. The stratified cross-validation ensures that the testing data in each fold is sampled from all classes. Our decision tree yields a high accuracy of 96.89%. Experiments on the classification model itself are presented in Section 4.

3.2.2. Workload Classification

The profiles of the workloads from the ALICE system collected in the previous step can be fed into a decision tree in real time. During classification, eight architecture-independent values in the workload profile is validated against each rule. The rules are obtained from linearizing the C4.5 algorithm decision tree during classification. If the condition is met for one of the seven rules, the Dwarf class is declared for that workload. The classification method can be illustrated in Algorithm 2.

3.3. Runtime Estimation Model

To estimate the runtime, we need to consider the machine architecture on which the workloads are run. Because our work seeks to predict the runtime of the workloads in the ALICE system that require high-performance computing (HPC), we focus on three (3) instance types that are chosen for HPC purposes

Algorithm 2: Workload Classification

```
Data: MICA metrics collected in the previous step
1 for rule 1 to rule 7 do
2     if data condition is met then
3         declare a Dwarf class
4         break
5     end
6 end
```

in Amazon EC2 [38], being general-purpose, compute-optimized, and memory-optimized instances. Consequently, we provide three (3) runtime prediction models for each dwarf (i.e., 21 runtime prediction models in total).

The runtime prediction model describes the relationship between the metrics, input size, and runtime. Both the metrics and runtimes can be obtained from MICA and `perf`. The input size can be obtained by normalization methods, as shown in Table 4.

Section 3.3.1 and 3.3.2 explain the details of this step for the Training and the Testing phases in our proposed framework, respectively.

3.3.1. Construction of the Runtime Estimation Model

To construct the runtime estimation model, we need to determine the relationship among the 12 metrics from MICA and `perf`, input size, and runtime of the workloads and then construct a set of equations that represent the relationships. We have limited the number of equation terms to not exceed 11 in order to control the number of possible equations. Each term can take the form of logarithmic, natural logarithmic, power, square root, or linear functions. Operation in an equation can either be '+' and '-'. Thus, the possible combination of equation terms can be as high as $13^{11} \times 5^{11} \times 2^{10}$ (13 possible parameters (12 metrics + input size); 5 possible functions for each term; 2 possible operations for each pair of terms). In order to select the equation that can best represent the relation of runtimes and its parameters, a Heuristic method is then required.

Table 4: Normalization of Input Size

Dwarf Class	Input Size	Remarks
dense	$n \times m$	n is the number of rows of a matrix/vector m is the number of columns of a matrix/vector
sparse	nnz	nnz is the number of non-zero elements
spectral	n	n is the number of data to be transformed
nbody	$n \times (time\ steps)$	n is the number of particles/bodies *time steps* is the number of time steps to be simulated
sgrid	$n \times m \times (time\ steps)$	n is the number of rows m is the number of columns *time steps* is the number of time steps to be computed
mapred	n	n is the number of data items
grapht	n	n is the number of nodes in a graph

Based on previous literature, the Artificial Bee Colony algorithm, also known as ABC, is our Heuristic method of choice.

ABC is an optimization algorithm that mimics the foraging behavior of bees. A set of feasible solutions to a problem is represented by the food sources. There are three types of bees in the hive: employed bees, onlooker bees, and scout bees. These bees iteratively perform different tasks for identifying food sources. The employed bees initially search for good food sources in the neighborhood. Once found, they will present qualities of their discovered food sources. The onlooker bees will forage in the vicinity of existing food sources presented by the employed bees. The best food sources have more possibility to be visited. This is the *exploitation* process, where the best among the neighbors is selected. On the other hand the food sources that are arid will be dropped and replaced by the new sources that are searched for by the scout bees. This process is the *exploration* process in the algorithm. The best food source will be kept in each iteration until the stopping criterion is met.

In our context, runtime estimation equations are the solutions and are represented as food sources. There are 3 types of bees iteratively perform different tasks for identifying the best estimation equation. According to Figure 3, the employed bees are responsible for the following tasks:

1. Randomly generating equation structures. For example, runtime $= \beta_1 x_1 + \beta_2 x_2 + ... + \beta_0$, where β_i and x_i are coefficients and independent variables, respectively.
2. Using the linear regression method to compute coefficients. The coefficients of the newly generated equation are unknown initially. Once proved that our collected data is normally distributed, linear regression was used to find the coefficients.
3. Computing R-squared [39] values of an equation. We compute R-squared in order to evaluate the accuracy, the prediction power for each randomly generated equation. The closer the R-squared value is to 1 (100%), the higher the accuracy of the prediction model.

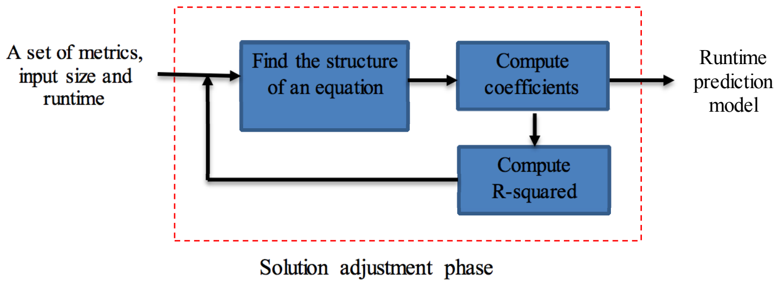
A set of metrics, input size and runtime
Find the structure of an equation
Compute coefficients
Compute R-squared
Runtime prediction model
Solution adjustment phase

Figure 3: Steps for the Artificial Bee Colony (ABC)

All the discovered equations from different employed bees are then sorted and given a probability based on the R-squared values. After that, each onlooker bee selects one of the structures based on the probability value and attempts to improve the structure. The structures that have no R-squared improvement for a certain period will be replaced by new structures that are generated by the scout bees. At the end of each iteration, the best equation structure and its coefficients are stored. The bees repeatedly improve the structures until the termination criteria is satisfied (the number of iterations reaches 10,000). In summary, the goal of ABC is to find the mathematical equation that can best describe the relationship among 12 metrics from MICA and perf, input size, and runtime.

For ABC, the solution is encoded in three main arrays: *Term*, *Function*, and *Operation*, as shown in Figure 4. As mentioned earlier, the search space for finding the equations can be as high as $13^{11} \times 5^{11} \times 2^{10}$. Due to this large search space, we adopt parallel computing [40] in order to improve the runtime performance of ABC. The algorithm ran on 12-core computers with 32 GB of memory. The number of bees (compute agents) used in our run was 3,600 in total (1,200 bees for each type of bee), and the algorithm ran until 10,000 iterations were completed.

Because ABC applies a heuristic method to search for a “good enough” solution in a limited amount of time, the best solutions from ABC may *not* be the same every time, even for the same training data. Consequently, we ran

ABC five times on each data set and selected the runtime equations with the highest R-squared value. Figure 5 shows the R-squared values of the runtime estimation equations obtained from our ABC. The R-squared values of nearly all the equations are higher than 90% for all of the dwarfs. This implies that ABC can efficiently find the model that can describe the relationship between the inputs and the runtime of a workload.

Note that our work focuses on applications where their behaviors fit in the context of a single dwarf. Applications whose behaviors span multiple dwarfs are out of the scope of this paper. To address this problem, however, we can add dwarf classes with mixed behaviors. For instance, workload classes would include the classes that represent the combinations of existing dwarf classes (e.g., *dense* & *sparse* class and *dense* & *grapht* class).

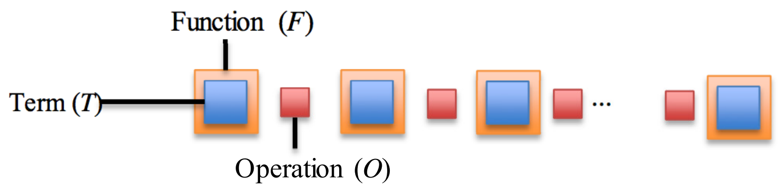

Figure 4: Structure of an ABC Solution

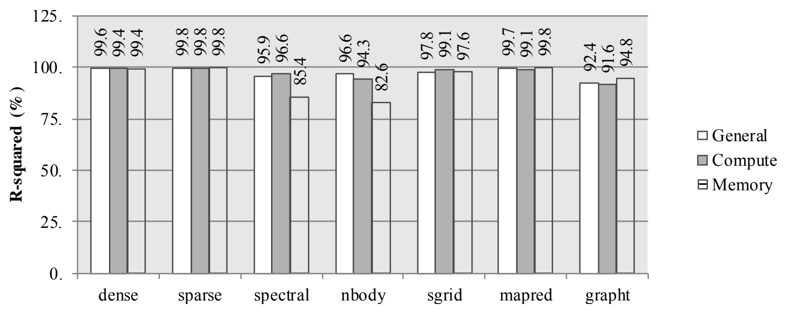

Figure 5: Percentage of R-squared values of Dwarfs on Virtual Machines

3.3.2. Runtime Estimation

To estimate the runtime of the workloads from the ALICE system, the prediction equation was selected from the 21 pre-generated equations based on the

Dwarf class (7 classes) that the workload belongs to and the HPC computing platforms (3 types of platforms) that the workload is executed on. The collected profile from the first step is then substituted in the equation terms and the runtime is computed.

3.4. Validation of Framework

This section seeks to validate the performance of each component in our proposed framework. Section 3.4.1 validates our proposed profile sampling. The correctness of our proposed workload classification is then validated in Section 3.4.2. Finally, Section 3.4.3 evaluates the performance of our proposed runtime estimation.

3.4.1. Validation of Profile Sampling

In our work, profiles of the test applications are created from a small window of execution time. This section presents results, which validate the fact that sample profiles can represent full run profiles relatively well and can thus be used to predict the runtime. We compared a sample profile against a full run profile for each benchmark by using the Z-score as a validation metric.

To reduce the computation time for the runtime estimation, we proposed collecting the profile of an application in the profile-sampling phase by using a sample datum, also called a "sample profile" (i.e., the applications sampling their profiles for a short period), instead of using the full-run data, also called the "full-run profile" (i.e., the profile collected from the beginning until the end of execution). This section seeks to verify that the sample data can be used instead of the full-run data for profile sampling. At the beginning, the profile of a benchmark was sampled by running it on the master computer for a short period (i.e., one minute since it was the lowest runtime in the training data).

In the experiment, we select two types of benchmarks, type I and type II, from each dwarf. The two benchmarks are the same application but with different input sizes. Type I and type II represent a small input size and a large input size, respectively. Table 5 shows the actual runtimes of the selected benchmarks. In the subsequent discussion, we compare the actual runtimes with the predicted runtimes.

Table 5: Actual Runtimes of Trained Benchmarks

Dwarf:Benchmark	Type	Actual Runtime (seconds)		
		General Purpose	Compute Optimized	Memory Optimized
dense:nn	I	3081	2492	1822
	II	15841	7572	5822
sparse:cg	I	233	159	140
	II	694	438	388
spectral:spectral	I	111	74	73
	II	286	186	177
nbody:nbody2d	I	894	785	592
	II	11291	9348	7020
sgrid:particle	I	2168	580	883
	II	96305	14174	26326
mapred:monteCarlo	I	3067	2022	1193
	II	12133	12457	2292
grapht:quickSort	I	370	245	219
	II	708	507	432

Before using the sample profiles to predict the runtime of the benchmarks, we plotted the Kiviat diagrams to determine the similarity between the sample data and the full-run data. The values plotted in the graphs are normalized as Z-scores.

For each diagram in Figure 6, the dashed line, which represents the sample data, nearly conceals the border of the grey area, which represents the full-run data. Thus, the sample data and the full-run data are approximately the same. Therefore, the sample data can be used to represent the full-run data and further be used in the model construction phase. However, the sample data should be

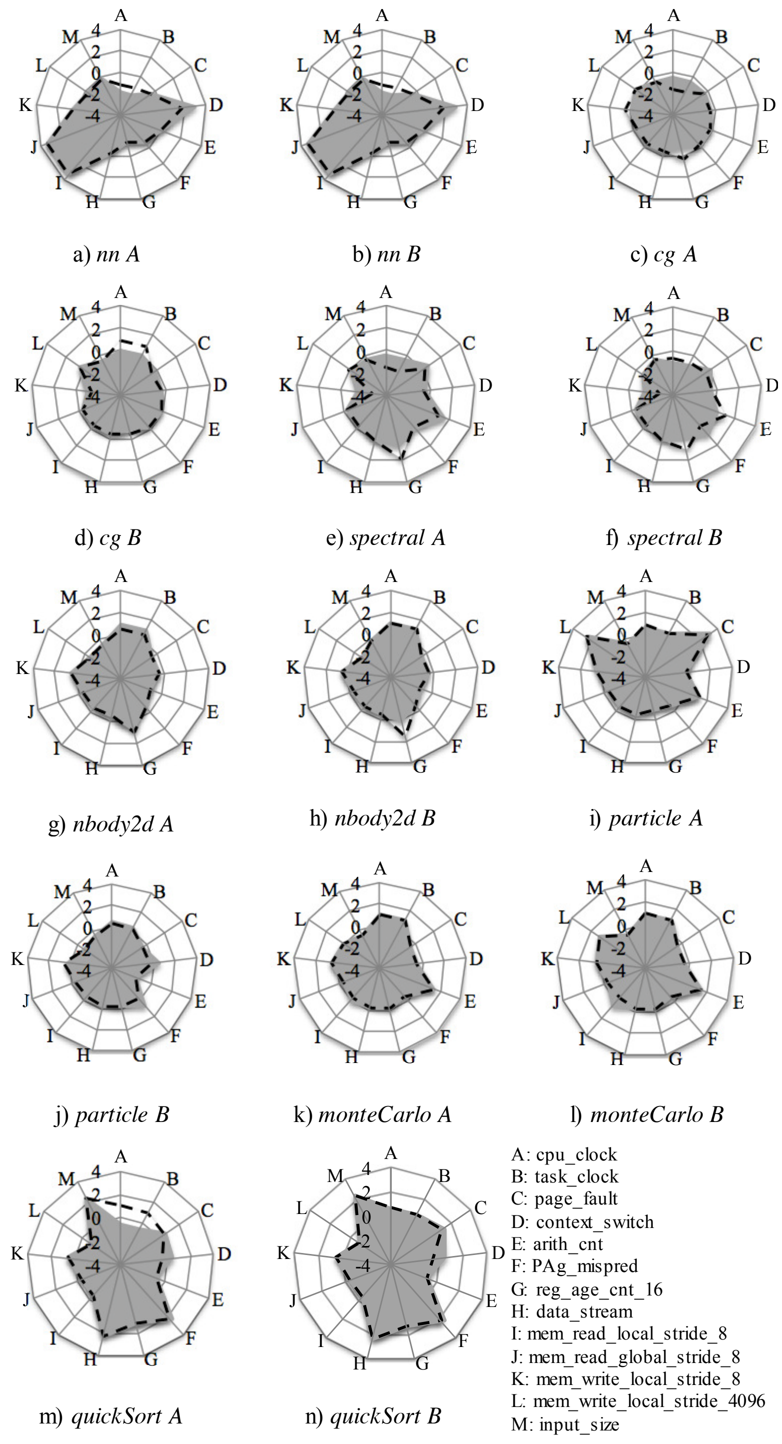

Figure 6: Kiviat Diagrams of Sample and Full-Run Data

used in the case that the training applications have long execution times, so it can substantially reduce the time required to train the models.

3.4.2. Validation of Workload Classification

This section validates that the C4.5 decision tree produces sufficiently good results for work-load classification. A set of labeled benchmarks are used to test the decision tree and the classification accuracy is measured.

To validate the classification correctness of our workload classification model, we use the model to predict classes of the trained benchmarks by using the sample data. The results show that all the benchmarks are correctly classified into their appropriate classes with the exception of *cg A*. The *cg A* benchmark actually belongs to *sparse*, but it is categorized as *sgrid*. (However, in the next step, we use both the sparse and *sgrid* runtime prediction models for *cg A*.)

3.4.3. Validation of Runtime Estimation

This section presents evaluation results of our runtime estimation. Three metrics are used: the prediction error percentage (EP), the mean absolute error percentage (MAEP), and the weighted absolute error percentage (WAEP). The runtime estimation model quality is evaluated, where the lower error percentages mean the better model quality.

Based on the results of the aforementioned workload classification, we select the appropriate model and use the aforementioned sample data for runtime prediction. The exception is the *cg A* application, where we leverage two models: *sparse* and *sgrid*. In order to evaluate the accuracy of the runtime prediction, we calculate a prediction error percentage (EP) of each data point using Equation 1.

$$EP = \frac{|A - P|}{A} \times 100 \tag{1}$$

We also calculate the mean absolute error (MAEP) [13] to evaluate the overall prediction error across all the benchmarks.

$$MAEP = \frac{\sum_{i=1}^{N} |A_i - P_i|}{N \times \sum_{i=1}^{N} A_i} \times 100 \tag{2}$$

With the same percentage of prediction error, the impact of the longer runtime jobs to the overall system is higher than the shorter ones. Thus, we calculate the weighted absolute error (WAEP) [41] in order to emphasize more on the impact of the errors of the long runtime jobs and less on the effect of the errors of the short jobs.

$$WAEP = \frac{\sum_{i=1}^{N} (|A_i - P_i| \times A_i)}{(\sum_{i=1}^{N} A_i)^2} \times 100 \tag{3}$$

Note that A is an actual runtime, P is a predicted runtime, N is the number of benchmarks that we want to take into account in MAEP or WAEP.

Table 6 presents the actual and predicted results as well as the prediction error percentages (EPs). Except for the *cg A* outlier, the maximum and minimum errors for the runtime predictions are 0.36% and 35.51%, respectively, which is better than what can currently be achieved via qualitative metrics such as the user name and project name in previous studies. Moreover, the mean absolute error percentage (MAEP) for each machine type, which is between 1.4% and 5%, suggests that the overall prediction result for all benchmarks is promising. The same applies to the weighted absolute error percentage (WAEP). However, WAEPs for the General Purpose and the Memory Optimized machines show that the major contribution of the errors comes from long runtime jobs because WAEPs are higher than MAEPs.

The three benchmarks that delivered runtime-prediction errors higher than 30% — *spectral A*, *nbody2d A*, and *quickSort A* — have short runtimes, i.e. less than 900 seconds in the training step. Because our framework is intended for HPC applications, which have significantly longer execution times, we expect that our models are more appropriate for predicting the runtime of such HPC applications.

For the *cg A* outlier, the framework mispredicted the runtime. The root cause for this misprediction still remains unknown, but as part of our future work, we seek to improve the robustness of classification and runtime prediction models and to use additional (and likely more diverse) data in the training step.

Table 6: Runtime Prediction Results for Trained Benchmarks

Benchmark	General Purpose			Compute Optimized			Memory Optimized		
	Actual (s)	Predicted (s)	EP	Actual (s)	Predicted (s)	EP	Actual (s)	Predicted (s)	EP
nn A	3081	3501	13.67	2492	3168	27.17	1822	2131	17
nn B	15841	15559	1.76	7572	7068	6.66	5822	5801	0.36
cg A on *sparse* model	233	253	8.71	159	208	31	140	145	3.73
cg A on *sgrid* model	694	5270	>100	438	27971	>100	388	5317	>100
cg B	694	758	9.4	438	495	13.17	388	384	0.77
spectral A	111	101	8.29	74	98	33.67	73	85	16.69
spectral B	286	278	2.54	186	196	5.84	177	203	15.06
nbody2d A	894	1062	18.83	785	839	6.91	592	800	35.17
nbody2d B	11291	12557	11.22	9348	9528	1.92	7020	8293	18.14
particle A	2168	2477	14.26	580	701	20.87	883	801	9.20
particle B	96305	76925	20.12	14174	16133	13.82	26326	18765	28.72
monteCarlo A	3067	3333	8.69	2022	2517	24.51	1193	1173	1.67
monteCarlo B	12133	9420	22.35	12457	8729	29.92	2292	2210	3.56
quickSort A	370	444	20.24	245	296	20.85	219	296	35.51
quickSort B	708	500	29.32	507	349	31.12	432	366	15.19
MAEP	1.34			4.61			2.05		
WAEP	8.80			3.57			9.25		

4. Runtime Estimation in the ALICE System

This section presents the performance of our framework in predicting the runtime of ALICE's applications. We focus on the scheduler for the offline applications run on the EPN cluster, where scientists from the ALICE collaboration often create and run new applications to analyze the collision data. In our experiments, the reference machine contained an 8-core Intel Core i7-2600 CPU, 8 GB of memory, and 470 GB of storage and ran the Scientific Linux CERN 6 (SLC 6) operating system.

To train the models, as outlined earlier in this paper, we collect the profiles of the benchmarks, shown in Table 3, by using MICA and perf tools on a reference machine. Because the execution times of ALICE's applications are relatively short, the time needed to construct the models using the full-run profiles is not measurably different from that of the sample profiles. Consequently, we used the full-run profiles to construct the models for runtime estimation. For each class of dwarf, we collected 15 profiles, where each profile contained 12 metrics — eight (8) from MICA and four (4) from perf. We then obtained 105 profiles of benchmarks to train the models.

For the workload classification model, we applied C4.5 to the training data in order to build a decision tree. The input attributes for the algorithm were only the eight (8) MICA metrics. Seven rules derived from the decision tree were used to determine the classes of applications. With stratified 10-fold cross-validation, our model can achieve 81.14% accuracy. The rules derived from the decision tree were used to categorize applications into a specific class.

In this test, we used four ALICE applications that run frequently in the EPN cluster to evaluate the performance of our framework. First, TPC-CE calibrates the central electrode of the Time-Projection Chamber (TPC) detector by analyzing ionization tracks left by a laser in the chamber. Second, PHS-GAIN measures the gain of the input channels of the PHoton Spectrometer (PHS) detector. This allows to adjust the bias of each APD (Avalanche Photo Diode) to have an equal gain. Third, SSD-PED measures the pedestal values

Table 7: Classification Results for ALICE's Applications

Application Name	Dwarf Class	MAEP	WAEP
TPC-CE	*dense*	1.02	1.24
PHS-GAIN	*sparse*	0.22	0.28
SSD-PED	*mapred*	0.28	0.26
MCH-PED	*spectral*	0.61	1

of the Silicon Strip Detector (SSD) detector channels, i.e. the value when no input signal is expected (empty event). This value can then be eliminated at runtime to reduce the data size by removing the constant and useless signal. Fourth, MCH-PED performs the same operation as SSD-PED but on the data of the Muon Chambers (MCH) detector, which has a different data format. We note that the execution patterns differ when running the same operations on the data from different detectors. Each of these applications creates statistics on a few hundred collision events, e.g., calculating an average value of a measured parameter.

To build a runtime prediction equation, we collected the full-run profiles of each application with various input sizes and used them to train the model. We constructed only models for the classes that the applications belonged to. From the classification rules, we could classify the applications into classes as shown in Table 7. Therefore, only *dense*, *sparse*, *mapred*, and *spectral* runtime prediction equations would be constructed.

We applied the Artifical Bee Colony (ABC) algorithm and linear regression on the collected data and derived the runtime equations, which each could yield at least 95% R-squared. The runtime equations for *dense*, *sparse*, *mapred*, and *spectral* are shown in Equations (4) through (7).

$$\begin{aligned}Runtime_{dense} &= 14 + 0.254\sqrt{Size_{Input}} + 5075P_{RegDist\leq16} + 60311P_{LRStride=0}\\ &+ 0.0441PF - 79824P_{LWStride\leq8} - 13.7\sqrt{CLK_{CPU}} + 153log(P_{LWStride\leq8})\\ &- 11.1\sqrt{PF} - 7104\sqrt{P_{LRStride=0}} - 1949Pct^2_{mult} - 146ln(WSS)\end{aligned} \tag{4}$$

$$\begin{aligned}Runtime_{sparse} &= -538.10 + 0.0175WSS + 943.1P^2_{LWStride\leq4096} + 7.944ln(CLK_{CPU})\\ &- 1.3634\sqrt{WSS} - 8.815(Size_{Input}) + 0.507(CS) + 65.514ln(B_{predict})\\ &+ 0.00012PF + 5.62ln(P_{LWStride\leq8}) - 0.0256\sqrt{PF}\end{aligned} \tag{5}$$

$$\begin{aligned}Runtime_{mapred} &= -4261 + 3.256\sqrt{Size_{Input}} + 246log(WSS) - 2.063ln(P_{GRStride\leq8})\\ &- 216.87ln(Size_{Input}) + 31.05ln(PF) + 0.1004Size_{Input} + 208.72ln(CS)\\ &- 0.00557WSS - 17.851\sqrt{PF} + 478.5ln(CLK_{Task}) + 1684\sqrt{P_{GRStride\leq8}}\end{aligned} \tag{6}$$

$$\begin{aligned}Runtime_{spectral} &= 56124 + 0.011\sqrt{Size_{Input}} - 38.69ln(Size_{Input}) + 12.08ln(PF)\\ &+ 17.05ln(WSS) + 145613P_{LWStride\leq4096} - 0.0114CLK_{CPU}\\ &+ 3.395ln(P_{LWStride\leq8}) - 30875P^2_{LWStride\leq4096} + 12844B_{predict}\\ &- 0.545\sqrt{CS} - 170519\sqrt{P_{LWStride\leq4096}}\end{aligned} \tag{7}$$

We then predicted runtimes for the ALICE applications with different input sizes. We calculated the error percentages (EPs) in the same fashion as for the previous experiment (see Equation 1). The runtime prediction results of TPC-CE, PHS-GAIN, SSD-PED, and MCH-PED are presented in Figure 7, Figure 8, Figure 9, and Figure 10, respectively. Please note that the labels on the graphs show the EPs.

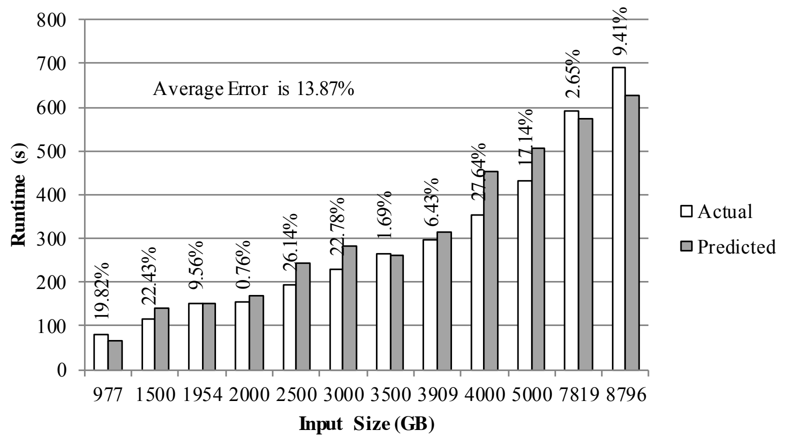

Figure 7: Runtime prediction results for TPC-CE (dense)

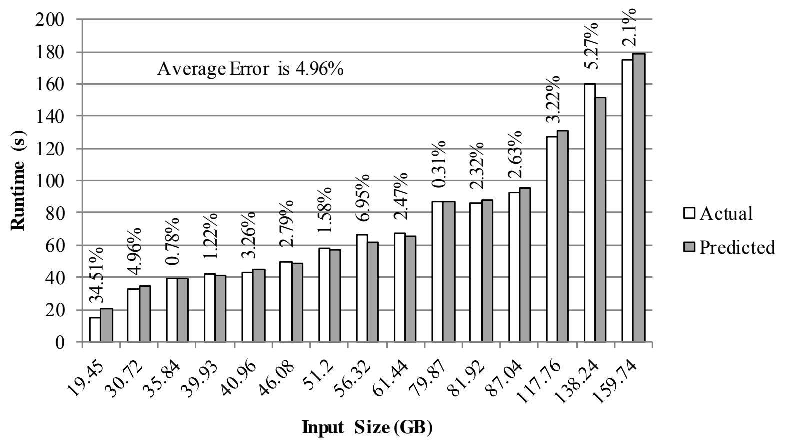

Figure 8: Runtime prediction results for PHS-GAIN (sparse)

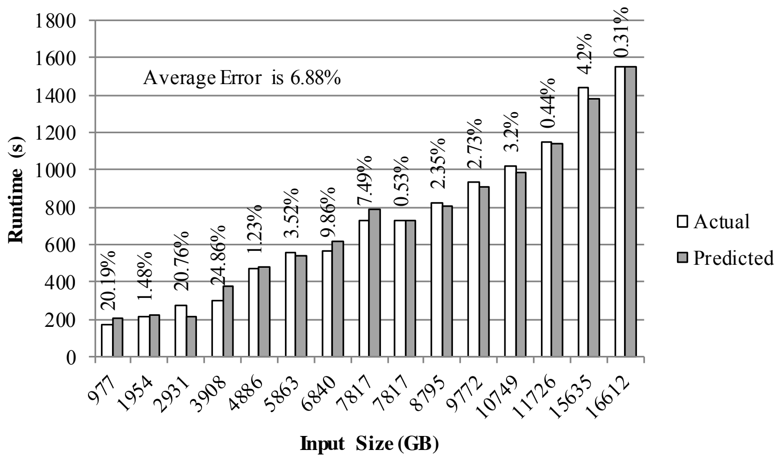

Figure 9: Runtime prediction results for SSD-PED (mapred)

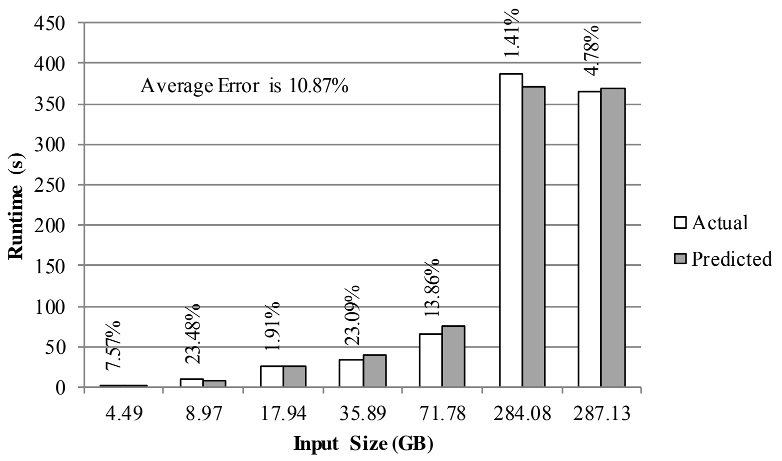

Figure 10: Runtime prediction results for MCH-PED (spectral)

Table 8: A Prediction Accuracy Comparison with Other Machine Specific Prediction Models

Technique(s) Used	MAEP
Various machine learning approaches & Regression [19]	$\leq$ 12%
Neural Network & Regression [20]	$\leq$ 10%
PQR2 [25]	$\leq$ 20%
Our Work: Decision Tree C4.5 & Regression	$\leq$ 5%

The runtime prediction result for each application was fairly accurate. The EPs are between 1% and 35%. Moreover, according to Table 7, the mean absolute error percentages (MAEP) and weighted absolute error percentages (WAEP) are below 2%.

To compare our runtime prediction performance with the previous works, we adopt the MAEP metric as it has been used across various previous works [13, 15, 16, 17, 19, 20, 22, 24, 25]. However, the data set in the experiments of previous works and our work differed. The comparison is drawn based on the assumptions that previous works have empirically selected the best experimental factors for their experiments. The goal of the comparison is to address that our proposed model has a comparable accuracy to the state of the art works. Since our proposed work is a Machine type specific prediction model, we compare the prediction performance with other Machine type specific approaches as shown in Table 8. We can see that our proposed work can provide a comparative MAEP to the previous works with $\leq$ 5%.

5. Discussion

In this section, we discuss some limitations of our framework and propose approaches to overcome such limitations in the future.

There are several factors causing the variation between predicted and actual runtime (e.g., network bandwidth, size of data, algorithms, and file dependency). Comparison between actual/predicted values should be controlled [42]. The practical physics applications used at CERNs ALICE are scheduled to be executed mostly on one machine. Therefore, in order to imitate real environment, we utilized a single ALICEs server in our experiment. Network bandwidth should not affect the runtime prediction. We have carefully controlled the machine specification for each prediction model. Consequently, the large discrepancy between our predicted runtimes and actual runtimes mainly results from the sizes of data and algorithms as follows:

1. When parameters (MICA/perf metrics and data size) and a runtime of an algorithm are not linearly correlated.
2. When a class of an algorithm is inconclusive (i.e. an algorithm is a combination of 2 classes or more).

All cases, when occur, can worsen the prediction accuracy. One way to improve the discrepancy is to generate hybrid-dwarfs and added them to the 7 dwarfs used in our work. The hybrid-dwarfs will cover more characteristics of applications. This is left for our future work.

Moreover, the accuracy could also be improved if a "white-box" approach was used. The "white-box" method can build a runtime estimation equation by using complexity analysis and the linear regression method where source codes of the applications must be given [43]. Although this method can provide higher accuracy, source codes of some applications cannot be provided. Also, this method requires a significant amount of manual processing. On the other hand, our proposed framework can be applied to applications, both without source codes ("black-box") and with source codes ("white-box"), to generate the runtime estimation equations with the same accuracy.

In fact, scientists at CERN create and run many testing applications in the EPN system on a regular basis in addition to the applications already in use. Consequently, the "white-box" approach would not be practical to manually create a runtime estimation model for every single application. For this reason, our runtime prediction mechanism for "black-box" applications is more practical for the EPN's scheduler.

6. Conclusion

Since the ALICE detector will be upgraded in 2018 to acquire more collision data, the scheduler for supporting the ALICE system has to be fast and highly efficient. One of the most important issues for the scheduler is how to accurately estimate the runtimes of the applications in the system because runtime is required by most scheduling algorithms. The main contribution of our work is a mechanism to estimate the runtimes of the applications with unknown profiles on the ALICE system. Our mechanism can support the workload scheduler that is practical and effective for particle physic studies in the near future. Similar to other runtime estimation approaches, our framework consists of two phases: workload classification and runtime prediction. However, the key attributes used in our framework are more informative than those of similar other works. We utilized 12 performance metrics, measured by the MICA and perf tools, rather than using the qualitative measures of a user name and a project name.

For workload classification, we realized a decision tree with the input of eight (8) MICA metrics. The output of the classification is one of seven (7) Berkeley Dwarfs classes. Each class has its own runtime estimation equation, where the model of the equation consists of the relationships among 12 performance metrics of MICA and perf, input size, and runtime of the workload. The Artificial Bee Colony (ABC) algorithm is then used to construct the runtime estimation model. However, the runtime equation is specific to the type of machine used.

We evaluated our framework by predicting the runtime of some of the ALICE applications. From the experimental results, the average runtime prediction accuracy for the ALICE system was approximately 90.85%. Therefore, our approach can efficiently estimate the runtime of the offline applications in the ALICE system and be further used to improve the scheduler performance in the EPN cluster of the ALICE system. In the future, we can extend our framework to provide APIs and runtime estimation service to typical schedulers used in HPC systems. In the framework extension, disaster recovery [44] and security of the scheduling node should also be considered for the ALICE system

Acknowledgements

The authors would like to thank the HRH Princess Sirindthorn Scholarship program, National E-Science Consortium, and CERNs ALICE O2 project for an opportunity in research collaboration and the internship at CERN Geneva.

References

[1] K. Aamodt, A. A. Quintana, R. Achenbach, S. Acounis, D. Adamová, C. Adler, M. Aggarwal, F. Agnese, G. A. Rinella, Z. Ahammed, et al., The alice experiment at the cern lhc, Journal of Instrumentation 3 (08) (2008) S08002.

[2] B. Abelev, A. collaboration, et al., Upgrade of the alice experiment: letter of intent, Journal of Physics G: Nuclear and Particle Physics 41 (8) (2014) 087001.

[3] A. A. D. P. Suaide, C. A. G. Prado, T. Alt, L. Aphecetche, N. Agrawal, A. Avasthi, M. Bach, R. Bala, G. Barnafoldi, A. Bhasin, et al., O2: A novel combined online and offline computing system for the alice experiment after 2018, Journal of Physics: Conference Series 513 (1) (2014) 012037.

[4] J. Shiers, The worldwide lhc computing grid (worldwide lcg), Computer physics communications 177 (1) (2007) 219–223.

[5] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, Characterization of backfilling strategies for parallel job scheduling, in: Parallel Processing Workshops, 2002. Proceedings. International Conference on, IEEE, 2002, pp. 514–519.

[6] W. Tang, N. Desai, D. Buettner, Z. Lan, Job scheduling with adjusted runtime estimates on production supercomputers, Journal of Parallel and Distributed Computing 73 (7) (2013) 926–938.

[7] K. Hoste, L. Eeckhout, Microarchitecture-independent workload characterization, IEEE Micro 27 (3) (2007) 63–72.

[8] perf manual, `http://man7.org/linux/man-pages/man1/perf.1.html`, accessed: 2016-11-27.

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al., The landscape of parallel computing research: A view from berkeley, Tech. rep., Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (2006).

[10] K. Vivekanandan, D. Ramyachitra, B. Anbu, Artificial bee colony algorithm for grid scheduling, J Converg Inf Technol 6 (7) (2011) 328–339.

[11] R. Baraglia, G. Capannini, M. Pasquali, D. Puppin, L. Ricci, A. D. Techiouba, Backfilling strategies for scheduling streams of jobs on computational farms, in: Making Grids Work, Springer, 2008, pp. 103–115.

[12] D. Tsafrir, Y. Etsion, D. G. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates, IEEE Transactions on Parallel and Distributed Systems 18 (6) (2007) 789–803.

[13] T. N. Minh, L. Wolters, Using historical data to predict application runtimes on backfilling parallel systems, in: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, IEEE, 2010, pp. 246–252.

[14] E. Gaussier, D. Glesser, V. Reis, D. Trystram, Improving backfilling by using machine learning to predict running times, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2015, p. 64.

[15] S. Krishnaswamy, S. W. Loke, A. Zaslavsky, Estimating computation times of data-intensive applications, IEEE Distributed Systems Online 5 (4) (2004) 127–136.

[16] W. Smith, I. Foster, V. Taylor, Predicting application run times with historical information, Journal of Parallel and Distributed Computing 64 (9) (2004) 1007–1016.

[17] E. Xia, I. Jurisica, J. Waterhouse, V. Sloan, Runtime estimation using the case-based reasoning approach for scheduling in a grid environment, in: International Conference on Case-Based Reasoning, Springer, 2010, pp. 525–539.

[18] Y. Zhang, W. Sun, Y. Inoguchi, Predict task running time in grid environments based on cpu load predictions, Future Generation Computer Systems 24 (6) (2008) 489–497.

[19] S. Kadirvel, J. A. Fortes, Grey-box approach for performance prediction in map-reduce based platforms, in: 2012 21st International Conference on Computer Communications and Networks (ICCCN), IEEE, 2012, pp. 1–9.

[20] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, T. Varvarigou, Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in cloud platforms, Future Generation Computer Systems 32 (2014) 27–40.

[21] R. Prodan, V. Nae, Prediction-based real-time resource provisioning for massively multiplayer online games, Future Generation Computer Systems 25 (7) (2009) 785–793.

[22] S. Islam, J. Keung, K. Lee, A. Liu, Empirical prediction models for adaptive resource provisioning in the cloud, Future Generation Computer Systems 28 (1) (2012) 155–162.

[23] J. Li, X. Ma, K. Singh, M. Schulz, B. R. de Supinski, S. A. McKee, Machine learning based online performance prediction for runtime parallelization and task scheduling, in: Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, IEEE, 2009, pp. 89–100.

[24] A. Matsunaga, J. A. Fortes, On the use of machine learning to predict the time and resources consumed by applications, in: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 495–504.

[25] D. Tetzlaff, S. Glesner, Intelligent prediction of execution times, in: Informatics and Applications (ICIA), 2013 Second International Conference on, IEEE, 2013, pp. 234–239.

[26] Y. Zhang, W. Sun, Y. Inoguchi, Predict task running time in grid environments based on cpu load predictions, Future Generation Computer Systems 24 (6) (2008) 489–497.

[27] A. Banharnsakun, T. Achalakul, B. Sirinaovakul, The best-so-far selection in artificial bee colony algorithm, Applied Soft Computing 11 (2) (2011) 2888–2901.

[28] D. Karaboga, B. Basturk, On the performance of artificial bee colony (abc) algorithm, Applied soft computing 8 (1) (2008) 687–697.

[29] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: artificial bee colony (abc) algorithm and applications, Artificial Intelligence Review 42 (1) (2014) 21–57.

[30] L. F. Manfroi, M. Ferro, A. M. Yokoyama, A. R. Mury, B. Schulze, A walking dwarf on the clouds, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, IEEE Computer Society, 2013, pp. 399–404.

[31] W.-c. Feng, H. Lin, T. Scogland, J. Zhang, Opencl and the 13 dwarfs: a work in progress, in: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ACM, 2012, pp. 291–294.

[32] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, K. Skadron, Rodinia: A benchmark suite for heterogeneous computing, in:

Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on, IEEE, 2009, pp. 44–54.

[33] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al., The nas parallel benchmarkssummary and preliminary results, in: Proceedings of the 1991 ACM/IEEE conference on Supercomputing, ACM, 1991, pp. 158–165.

[34] A. Kaiser, Torch computational reference kernels-a testbed for computer science research, 2011.

[35] K. Manakul, P. Siripongwutikorn, S. See, T. Achalakul, Modeling dwarfs for workload characterization, in: Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on, IEEE, 2012, pp. 776–781.

[36] U. Taetragool, T. Achalakul, Method for failure pattern analysis in disk drive manufacturing, International Journal of Computer Integrated Manufacturing 24 (9) (2011) 834–846.

[37] K.-Y. Chou, C.-C. Shih, H.-C. Keh, P.-Y. Yu, Y.-C. Cheng, N.-C. Huang, Using decision tree to analyze patient of aortic aneurysm with chronic diseases in clinical application, in: 2013 16th International Conference on Network-Based Information Systems, IEEE, 2013, pp. 405–409.

[38] A. E. C. Cloud, User guide, api version jun. 15, 2014.

[39] D. C. Montgomery, Design and analysis of experiments, John Wiley & Sons, 2008.

[40] W. Gropp, E. Lusk, N. Doss, A. Skjellum, A high-performance, portable implementation of the mpi message passing interface standard, Parallel computing 22 (6) (1996) 789–828.

[41] S. Vasupongayya, S.-H. Chiang, Performance problems of using system-predicted runtimes for parallel job scheduling, in: Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems, ACTA Press, 2007, pp. 369–374.

[42] V. Chang, G. Wills, A model to compare cloud and non-cloud storage of big data, Future Generation Computer Systems 57 (2016) 56–76.

[43] S. Pumma, T. Achalakul, L. Xiaorong, Automatic vm allocation for scientific application, in: Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on, IEEE, 2012, pp. 828–833.

[44] P. Mejía-Alvarez, D. Mossé, A responsiveness approach for scheduling fault recovery in real-time systems, in: Real-Time Technology and Applications Symposium, 1999. Proceedings of the Fifth IEEE, IEEE, 1999, pp. 4–13.