THE RUNTIME ESTIMATION FRAMEWORK FOR BLACK BOX SCIENTIFIC APPLICATIONS

MISS SARUNYA PUMMA

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF ENGINEERING (COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2013

The Runtime Estimation Framework for Black Box Scientific Applications

Miss Sarunya Pumma B.Eng. (Computer Engineering)

A Thesis Submitted in Partial Fulfillment
of the Requirement for
the Degree of Master of Engineering (Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2013

Thesis Committee

…… (Asst. Prof. Marong Phadoongsidhi, Ph.D.)	Chairman of Thesis Committee
……. (Assoc. Prof. Tiranee Achalakul, Ph.D.)	Member and Thesis Advisor
……. (Asst. Prof. Santitham Prom-on, Ph.D.)	Member
……. (Phond Phunchongharn, Ph.D.)	Member
……. (Pierre Vande Vyvre, Ph.D.)	Member

Thesis Title	The Runtime Estimation Framework for Black Box Scientific Applications
Thesis Credits	12
Candidate	Miss Sarunya Pumma
Thesis Advisor	Assoc. Prof. Dr. Tiranee Achalakul
Program	Master of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2013

Abstract

The growth of data and computation in the past decade has brought about the needs for cloud infrastructure. Cloud leverages the Internet as a tool through which remote computers can share resources on-demand. The cloud infrastructure can be utilized as a high performance computing (HPC) platform which contains flexible and excessive computing resources. In order to efficiently run the HPC applications in the cloud, a great deal of technical knowledge is required. One of the challenges is how to estimate the runtimes of applications accurately because an inaccuracy in runtime estimation can lower the overall performance of a computer system. Moreover, runtime is an important attribute for tasks scheduling. For instance, most well known scheduling algorithms, such as, Backfilling and Heterogeneous Earliest Finish Time (HEFT), use runtime to determine the schedule of the tasks. In this thesis, we have proposed a runtime estimation method for unknown-profile applications in the cloud computing environment. Unlike other approaches, we also provide a procedure to collect the profiles of applications, which are the metrics that represent the execution behavior of an application. This allows our approach to predict the runtime of the HPC applications even if the metadata is not provided. In order to predict a runtime of a workload, only two steps are required. In the first step, the application will be classified into a class based on the similarity of the execution characteristics. In our work, we have adopted the Berkley's Dwarfs taxonomy to define the classes. The classification result will be used to choose a runtime prediction equation for the workload. In the next step, the runtime will be predicted by using the equation that is selected in the previous step. The runtime prediction equations are constructed by using the Artificial Bee Colony (ABC) and the linear regression techniques. In order to verify the practicality of our framework, we predicted the runtimes of the HPC applications on three types of virtual machines, General purpose, Compute Optimized, and Memory Optimized instances, provided by Amazon EC2. Our method can yield low prediction error percentages in most cases. Moreover, it can provide more accurate runtime prediction results in comparison to the user-estimation method.

Keywords : Amazon EC2 / Berkley's Dwarfs / Cloud Computing / Job Scheduling / Runtime Estimation

หัวข้อวิทยานิพนธ์	การคำนวณเวลาในการประมวลผลแอพพลิเคชันเชิงวิทยาศาสตร์
หน่วยกิต	12
ผู้เขียน	นางสาวศรัณยา ภุมมา
อาจารย์ที่ปรึกษา	รศ. ดร. ธีรณี อจลากุล
หลักสูตร	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	วิศวกรรมศาสตร์
ปีการศึกษา	2556

บทคัดย่อ

อัตราการเพิ่มขึ้นของข้อมูลและการประมวลผลในช่วงสิบปีที่ผ่านมานำมาซึ่งความต้องการในการใช้บริการการประมวลผลแบบกลุ่มเมฆหรือคลาวด์ ทั้งนี้ระบบคลาวด์ใช้ระบบอินเตอร์เน็ตเป็นเครื่องมือในการรวมทรัพยากรคอมพิวเตอร์ไว้ด้วยกันเพื่อตอบสนองความต้องการของผู้ใช้ โดยระบบคลาวด์สามารถถูกนำมาใช้เป็นแพลตฟอร์มสำหรับการประมวลผลแอพพลิเคชันที่ต้องการทรัพยากรในการประมวลผลสูงได้ เนื่องจากผู้ใช้สามารถร้องขอทรัพยากรในการประมวลผลหรือจัดเก็บข้อมูลได้ตามความต้องการ อย่างไรก็ตามการประมวลผลบนคลาวด์ต้องอาศัยความรู้เชิงเทคนิคอย่างมาก ซึ่งหนึ่งในปัจจัยที่ควรจะต้องพิจารณาคือวิธีการที่มีประสิทธิภาพในการคำนวณเวลาที่ใช้ในการประมวลผลแอพพลิเคชัน เนื่องจากความคลาดเคลื่อนในการคาดการณ์เวลาที่ใช้ในการประมวลผลอาจส่งผลต่อประสิทธิภาพโดยรวมของระบบคอมพิวเตอร์ได้ นอกจากนี้ตัวจัดลำดับงาน (Scheduler) ที่เป็นที่นิยม เช่น Backfilling และ Heterogeneous Earliest Finish Time (HEFT) ใช้เวลาในการประมวลผลเป็นตัวแปรที่สำคัญในการจัดลำดับงานภายในระบบ ดังนั้นงานวิจัยนี้จึงนำเสนอวิธีการในการคำนวณเวลาในการประมวลผลสำหรับแอพพลิเคชันเชิงวิทยาศาสตร์บนระบบคลาวด์ โดยแอพพลิเคชันดังกล่าวมักจะต้องการทรัพยากรในการประมวลและจัดเก็บข้อมูลจำนวนมาก ทั้งนี้วิธีการที่นำเสนอครอบคลุมไปถึงวิธีการเก็บโปรไฟล์ (Profile) ของแอพพลิเคชันซึ่งถูกจัดเก็บในรูปแบบของเมตริก (Metrics) ซึ่งใช้ในการอธิบายลักษณะเฉพาะในการประมวลผลของแอพพลิเคชันนั้นๆ ทั้งนี้การเก็บโปรไฟล์ของแอพลิเคชันจะทำให้สามารถคาดการณ์เวลาในการประมวลผลได้โดยไม่ต้องอาศัย เมตาดาต้าของแอพพลิเคชัน (Metadata) ซึ่งการคำนวณเวลาในการประมวลผลสำหรับงานวิจัยนี้ประกอบด้วย 2 ขั้นตอนหลัก โดยในขั้นแรกแอพพลิเคชันจะถูกแบ่งออกเป็นคลาส (Class) โดยพิจารณาจากลักษณะเฉพาะในการประมวลผล ทั้งนี้คณะนักวิจัยได้อ้างอิงการแบ่งคลาสตามงานวิจัยของมหาวิทยาลัยแคลิฟอร์เนีย เบิร์กลีย์ ซึ่งแบ่งแอพพลิเคชันเชิงวิทยาศาสตร์ออกเป็นหมวดหมู่ซึ่งถูกเรียกว่า คนแคระ (Dwarf) ตามลักษณะการประมวลผลของแต่ละแอพพลิเคชัน ทั้งนี้ผลการแบ่งคลาสจะถูกนำไปใช้ในการเลือกสมการในการคำนวณเวลาในการประมวลผลของแอพพลิเคชันในขั้นตอนต่อไป สมการดังกล่าวถูกสร้างขึ้นโดยใช้อัลกอริทึมการจำลองฝูงผึ้ง (Artificial Bee Colony หรือ ABC) และการวิเคราะห์การถดถอยเชิงเส้น (Linear Regression) สำหรับการประเมินประสิทธิภาพของวิธีการที่นำเสนอนั้น คณะนักวิจัยได้นำวิธีดังกล่าวไปใช้

ในการคำนวณเวลาในการประมวลผลของแอพพลิเคชันต่างๆ บนเครื่องคอมพิวเตอร์เสมือน (Virtual Machine) 3 รูปแบบซึ่งให้บริการโดยอเมซอนอีซีทู (Amazon EC2) คือ เครื่องแบบทั่วไป (General Purpose) เครื่องสำหรับการคำนวณที่ใช้หน่วยประมวลผลสูง (Compute Optimized) และเครื่องสำหรับการคำนวณซึ่งใช้หน่วยความจำสูง (Memory Optimized) ทั้งนี้วิธีการที่นำเสนอสามารถคาดการณ์เวลาที่ใช้ในการประมวลผลด้วยความคลาดเคลื่อนต่ำ รวมทั้งยังมีความแม่นยำมากกว่าวิธีที่ให้ผู้ใช้เป็นผู้ประมาณเวลาในการประมวลผลอีกด้วย

คำสำคัญ : การคำนวณเวลาในการประมวลผล / การจัดลำดับงาน / คนแคระของเบิร์กลีย์ / ระบบคลาวด์ / อเมซอนอีซีทู

ACKNOWLEDGEMENTS

This project could not have been successful without the help of my advisor, Assoc. Prof. Dr. Tiranee Achalakul. She has given me a great deal of useful advice as well as support throughout. The motivation she gave me has driven me to work hard on this project. I would like to thank her for being the guiding light when I was lost. Without her, this project would have been a painful ordeal.

I would like to express my sincere gratitude to Mr. Kittituch Manakul. He was always my mentor who helped me from the beginning to the end of this project. Comments and feedbacks from his own experience and his broad knowledge were extremely useful for me. Apart from technical and theoretical supports he gave, he was also an emotional supporter. Thank you very much for helping me through the tough times.

I also would like to express my appreciation to Mr. Sylvain Chapeland, my mentor at CERN, who provided the computer facilities as well as the applications for me to run the experiments at CERN. Besides, he gave such valuable advice on how this project could nicely fit in the ALICE O2 project.

Moreover, I would like to thank the network laboratory's administrators: Mr. Nipat Sukittiwong, Mr. Warat Puengtambol, and Mr. Chaiwat Kaewyai. They kept and maintained the computer resources in perfect shape for the experiments that could run with almost zero downtime.

Furthermore, I would like to state that this project was funded by the Thailand Research Fund-Master Research Grant (TRF-MAG) in collaboration with the Venture Catalyst Company. The computer facilities, equipment, and cloud computing costs were supported by this grant.

Lastly, I would like to thank you my family and my lab mates at the Concurrent Algorithms and Scalable Technology Lab (CAST Lab) who always cheered me up when I was frustrated. Without these people, obstacles and problems would have been impossible to solve.

CONTENTS

PAGE

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

SYMBOL		UNIT
<	Less than sign	-
>	Greater than sign	-
≤	Less than or equal sign	-
≥	Greater than or equal sign	-
^	And operation	-
∨	Or operation	-
I/O	Input and output	-
R-squared	Coefficient of determination in statistics	percent
Z	Standard score in statistics	-

LIST OF TECHNICAL VOCABULARY AND ABBREVIATIONS

ABC	=	Artificial Bee Colony
ALICE	=	A Large Ion Collider Experiment
Amazon EC2	=	Public cloud service provided by Amazon.com
CERN	=	The European Council for Nuclear Research
Cloud	=	An emerging computing platform with on-demand services
CPU	=	Central processing unit of a computer
DAQ	=	Data acquisition
Dwarf	=	Computational kernel representing a type of execution pattern for scientific applications
GA	=	Genetic Algorithm
GPU	=	Graphics processing unit of a computer
HEFT	=	Heterogeneous Earliest Finish Time
HPC	=	High performance computing
K-NN	=	k Nearest Neighbor
MICA	=	Microarchitecture-Independent Workload Characterization

CHAPTER 1 INTRODUCTION

At present, cloud computing is getting more and more popular because of its ability to provide the user with resources automatically in a short amount of time. Since cloud combines computer resources, such as CPU, memory, and storage, into a virtual place called a resource pool, a size of computer resources is changeable depending on requirements of user. Moreover, the computational resources can be accessed from anywhere via the Internet and are charged as utilization rate of computing power, bandwidth, and storage. For these reasons, cloud computing can satisfy the immediate need of low cost computing resources.

The cloud platform can be categorized into three main models: public cloud, private cloud, and hybrid cloud. A public cloud provides various types of service models that charge per usage. Thus, the cost for a use of a public cloud is relatively low because there is no purchasing of machines and no maintenance costs. However, an execution performance of public cloud cannot be controlled because the computer resources that are provided in the public cloud are not 100% dedicated to a single user. Anyone who has a user account can access the public cloud's network via the Internet. On the contrary, a private cloud entails high costs for machine purchasing and maintenance. Hybrid cloud, which is a combination between public and private clouds, is thus introduced in order to gain more benefits from both types of cloud to improve the execution performance and cut down the costs.

All the cloud computing platforms mentioned above could be utilized as the high performance computing (HPC) platforms that support scientific application executions. The scientific applications, which can be categorized into the compute-intensive application and the data-intensive application, usually consume a great deal of computer resources and execute for a long period. The benefit of using the cloud is to improve the performance of execution with regards to optimizing the execution time and costs.

To run the HPC applications in the cloud environment efficiently, a smart job scheduler is required. Most scheduling algorithms need the runtimes of the jobs in order to assign them to execute on the appropriate machines. For example, Backfilling [1] and Heterogeneous Earliest Finish Time (HEFT) [2] require runtime in their scheduling processes. Thus, the application runtime is an important attribute for the scheduler. However, predicting the runtimes of the applications in a computer system is a difficult task because much technical knowledge is needed. To avoid the difficulty, some computer systems ask the runtime from the user. Although this method is easy, it is inaccurate and inefficient because the user almost always overestimates the runtimes of their applications [3]. In addition, some existing runtime estimation methods assume that the same user usually runs the same application in the system [4]. 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, this assumption can be violated easily because it is not always true that the same user will run the same jobs every time. Moreover, the key used in this method is not informative. It does not contain any running behavior of the application.

We then are proposing a methodology to estimate the runtime of the applications in the cloud computing environment efficiently. Unlike other approaches, we use the informative application characteristics, which contain the execution behavior, to predict

a runtime of the application. When the application is submitted to the system, these characteristics will be sampled by using the workload characterization tools, namely Mica and Perf. Our framework consists of two components: the workload classification model and the runtime prediction model. These two components work in sequence. The input of both components is a set of MICA and Perf metrics. The workload classification model is responsible for categorizing the application into a certain class based on its characteristics. The characteristics of the applications in each class are referred from the Berkley's Dwarfs taxonomy. In our work, we have a separate runtime prediction model for each class of applications on a specific type of virtual machines. To select the appropriate model for the application, the classification result is used. The prediction model is a mathematical model, built by using the Artificial Bee Colony (ABC) optimization with the linear regression technique. With our framework, the runtime of the workload can be simply predicted without any additional information about the application.

This report is structured as follows: Section 1 has stated introduction of the study, Section 2 presents the related works and the background study along with the discussions, Section 3 describes our proposed runtime estimation methodology, Section 4 shows the experiments and results, and Section 5 is the conclusion.

CHAPTER 2 RELATED WORKS AND BACKGROUND STUDIES

In this section, a survey of the related works and background knowledge is presented. The first sub-section gives details of related works. The other section provides information on topics and techniques that have been adopted in our framework.

2.1 Related Works

The runtime of the application is an important attribute in most scheduling schemes. Several scheduling algorithms, for example, Backfilling, and Heterogeneous Earliest Finish Time (HEFT), use the runtime in order to optimize the execution costs, such as, execution time, monetary cost, and energy consumption. Backfilling [1] is one of the First Come First Serve (FCFS) algorithms that allow lower priority applications to be scheduled while the first job in a queue (highest priority) is waiting for the computing resource. Since Backfilling has to fill short jobs in the available slots without delaying high priority jobs, the runtimes of the applications are required. HEFT [2] is a well-known workflow-scheduling algorithm that schedules tasks based on priorities. Similar to the Backfilling algorithm, HEFT needs the runtimes in order to calculate the priorities of nodes in the workflow. Without the accurate runtimes, the scheduling results might not be satisfactory.

In a computer system, the runtimes of the applications, which are submitted to the system, is usually provided by users. However, the user almost always overestimates the runtimes, which can violate the overall performance of the system [3]. Tang et al. [4] analyzed the job trace from Argonne's Intrepid and found that there were about 50% of 275,000 jobs in the system that used only half of the user-estimated runtime to finish their executions. They then proposed a methodology to adjust the user-provided runtime in order to improve the performance of the supercomputer system. They determined the accuracy of the user-estimated runtime by using $R = t_{act}/t_{user}$, where t_{act} is the job's actual runtime and t_{user} is the user-estimated runtime. The range of R was between 0 and 1. If R was close to 1, then the user-provided runtime was highly accurate. The equation to adjust the runtime was simple: $t_{schd} = t_{user} \times A$, where t_{schd}, t_{user}, and A were the time used by the scheduler, the time provided by user, and the adjustment factor, respectively. In order to determine the value of A, four runtime adjustment schemes were presented. The adjustment schemes searched for the similar application based on keys (user name of the user who submitted jobs and project name) from the historical data in the limited time frame and calculated the value of A by averaging the R-values of the similar applications. This proposed work could significantly improve the runtime and the overall performance of the system. However, the method just assumes that the same user will always submit the same project to the system. This assumption cannot be applied in the cloud environment since it cannot cover all the usage scenario of the cloud. For example, if user A employs the public cloud resources to run different tests, the runtime of any applications submitted by user A cannot be predicted. Moreover, the attributes used in the runtime prediction do not actually represent the characteristics of the application. To solve this, more informative attributes are needed.

Krishnaswamy et al. [5] proposed a method to estimate the computation times for data-intensive applications. Rather than using the user name and the project name to determine the similarity between two applications, they adopted the rough sets theory, which could handle the uncertainty in data. The rough sets theory used the historical

data to find the subset of attributes that strongly related to the runtimes. The output of rough sets was a similarity template. A submitted application would be compared to the historical data using the similarity template. Then, the runtime for the application would be a mean of the runtimes of the applications in the same category. The work could yield a high accuracy for the data mining and high-performance computing applications. Nevertheless, this work did not explicitly present the set of attributes that were used in the similarity template building so the runtime prediction results may not be precise, in the case that the historical data of some applications are limited. In other words, the results can only be good in the scenario where most applications are repeatedly executed.

In line with previous work, Smith et al. [6] implemented their runtime prediction framework based on the similarity templates of the historical applications. The template was used to classify applications into groups. The attributes used for classification were a set of workload characteristics obtained from the computer system. The search techniques, which are greedy and genetic algorithms, were applied to the framework for determining the similarity template. The runtime of the application could be derived in two ways: using the mean of the runtimes or using the linear equation to calculate the runtime. The linear equation used in this work simply established the relationship between runtime and number of compute node requested by user: $runtime = aN + b$, where a and b were the coefficients. Notice that the coefficients in linear equations were different among different categories. In this work, the application might belong to more than one category. Therefore, the estimated runtime with a smallest value of confidence interval was selected as an application runtime. Since different computer systems may contain different sets of characteristics, the algorithm cannot guarantee that the prediction performance of the framework will be good if certain attributes are missing.

The case-based reasoning approach was presented in the work of IBM Canada Lab. Xia et al [7] calculated the runtime from the historical information stored in the form of cases. In this work, the cases were used to determine the similarity between the applications and computers. The cases were defined by using the TA3 algorithm, which was a case-based reasoning approach. TA3 classified the cases based on the similar runtime assumption determined by a standard deviation. For example, two applications that had the similar runtimes on the similar machines would be considered to be similar. A case was represented in a data record that contained the job and machine characteristics and a priority of the case. The cases priority was used to select the runtime of the application in the case that the application was categorized into many cases. To estimate the runtime of the new application, the k-nearest neighbor algorithm was adopted. It categorized the application into the groups of applications that provide the smallest Euclidean distance. Then, the runtime was the average value of the runtimes in the case. This approach was experimented on a real system of IBM to schedule Functional Regression Tests (FRT), which required a frequent test on various platforms. It could obtain high runtime estimation accuracy and achieve higher system performance. One drawback of this approach is the certain number of cases is not predefined. The number of cases may grow without a boundary, which can deteriorate the performance of the system. Therefore, the policy to control the number of cases must be well defined.

The previous works used the application-oriented approach that directly employed the application information, i.e. user name and project name, for runtime prediction. Zhang

et al. [8] proposed a different method, which is the resource-oriented approach, to predict the runtime of the workloads in a grid environment. The resource-oriented method estimated the runtime by adopting the prediction information of the future resources allocation. This approach exploited the benefit of grid that the information of resources could be obtained from Grid Information System (GIS). Only the information of the CPU load was used in this work. Therefore, this framework proposed a method to predict the number of CPU loads of the applications in grid. The complex time series model was adopted in CPU loads prediction. Once the number of CPU loads was obtained, it was fed to the mathematical model to predict the estimated runtime. The simulation of the framework showed excellent runtime prediction results in the grid environment. However, this method cannot be applied to the cloud environment since the cloud does not allow users to access any hardware event.

From all the related works, the general model for runtime prediction approaches comprises 2 main parts: workloads similarity identification and runtime estimation. Most approaches adopt the classification or clustering technique in applications similarity determination. However, certain sets of attributes used in classification or clustering are not explicitly defined in some approaches. This can result in missing of data attributes and subsequently can affect to the performance of workload similarity identification and runtime prediction. Therefore, a common attributes set, which can be collected by using standard method, should be defined. Moreover, some attributes, for example, user name and project name, do not have direct impact on the runtime of the application. This kind of attributes can be used only in the case that users always run the same application in the system. Therefore, the set of attributes or metrics should actually represent the characteristic of the application, for example, data movement pattern and execution behavior.

In the runtime estimation step, a mean of the runtimes of the similar applications is generally used. It is easy to be determined, but may not be precise. As mentioned above, a user name and a project name do not truly relate to a runtime. Therefore, the average runtime of the application submitted by the same user with the same project name cannot ensure that the runtime of the applications will be the same.

Similar to the general runtime prediction approach, our runtime estimation framework is also divided into two main parts: workload similarity determination and runtime estimation. In workload similarity determination, we employed a set of performance metrics, which can be collected by the MICA [13] and the Perf tools. These metrics are allowed to be collected on the cloud platform. They can capture the execution behavior of the applications, of which data is more informative than attributes used in the related works. The metrics are used to categorize an application into a certain class. We used the taxonomy of Berkley's Dwarfs [14] to define the classes. The taxonomy categorizes the high performance computing applications into 13 classes based on the data transfer and running patterns. However, we implemented only seven classes of the dwarfs because we cut out some classes that give the redundant characteristic. More detail will be provided in the next subsection.

In our work, we used the linear regression equations to estimate the runtime of the application. We built a separated runtime prediction model for each class of dwarfs and for a specific type of virtual machines in the cloud environment. Therefore, there were seven possible runtime prediction models for the application to select. The classification

information from the previous step is used to select the runtime estimation model. For example, if the runtime of the application *A* on the machine *M* is required and the application *A* is classified into class *C*, then the runtime estimation model for class *C* on machine *M* is selected.

Our methodology can be used to predict the runtime of the application on the virtual machines in the cloud environment efficiently. Since we also provide the method to collect the data attribute used in runtime prediction, the problem of attribute missing can be solved. Moreover, we used the more informative data attributes in application similarity identification step. Therefore, this framework can be used to predict the runtime of the application without having to know any metadata of the workload.

2.2 Background Study

In order to implement the mechanism to predict the runtime of the scientific application in the cloud environment, we need to review the Amazon's cloud services, the application's performance measurement tool called Microarchitecture-Independent Characterization or MICA, the 13th Berkley's dwarfs, and the optimization technique used in model construction phase. Details of each study will be expressed and discussed one by one. Moreover, the method and its package that were adopted in our work will be presented in the sub-sections below.

2.2.1 The Public Cloud: Amazon Elastic Compute Cloud

The purpose of this project was to estimate the runtime of the application on the cloud. The services on the public cloud computing were then studied.

Amazon Web Service (AWS), Microsoft Azure, Rackspace, and Newservers are a few of well-known cloud service providers. These cloud providers offer the infrastructure as a service (IaaS) to the users. NewServers is the only one that provides physical cloud computing while others offer the virtual machines. Platform Computing Corporation [9], an IBM company, tested and measured the quality of each cloud provider in many aspects against the standard benchmarks. The instances that are likely to be the user's choice for running high performance computing (HPC) applications were used in the experiments. From the testing results, it turned out that Amazon EC2's instances outperformed others in almost all benchmarks. Even though the price is the most expensive, it is worth of investment [9].

In public cloud, the computing performance is unpredictable because a physical machine is shared among users. According to the testing results of Platform Computing Corporation mentioned in the previous paragraph, the public cloud that we concerned is Amazon EC2. The instance type of Amazon EC2 can be divided into six categories [10]; standard instances, micro instances, high-memory instances, high-CPU instances, cluster compute instances, and cluster GPU Instances. Each type of instances has different capacities – memory, CPU (EC2 compute unit), storage, I/O performance, and CPU utilization percentage. Details on each type of instances are as follows:

The *Standard instances* provide a proper proportion between memory and CPU for general application, and they limit the CPU utilization to be a maximum at 50 percent for a single processor core [11]. The *high-memory* and *high-CPU instances* have high capability specifically for high throughput applications and compute-intensive applications, respectively. The *Micro instances* are suitable for an application that does

not require high throughput because EC2 computes units can burst for a short period which means they allow high CPU utilization percentage only in a short period. The *Cluster Compute instances (CCI)* are suitable for HPC application because of a large number of computing units and high network performance. The *Cluster GPU instances* are like having extra graphics processing units (GPUs) attached to the Cluster Compute instances, therefore; they are well suited for HPC applications and also media processing applications.

In general, cloud service providers offer two purchasing alternatives for cloud instances; on-demand instance and reserved instance [12]. The on-demand instances are charged according to the per-hour basis usage rate while another instance type uses the advance payment policy. The on-demand instances will be more convenient but more expensive than the reserved instances. However, users will need a good plan to utilize the reserved instances.

The virtual machine instances used in our platform are the high-memory, the high-CPU, and the general-purpose instances in the Amazon Web Service.

2.2.2 Microarchitecture-Independent Workload Characterization: MICA

In order to estimate the runtime of an application, its profile is needed to be collected. In this work, we adopted MICA metrics as the application's attributes that will be used in runtime prediction.

The MICA [13] project is the work of Hoste and Eeckhout from Ghent University. MICA is a pin tool, which is a computing analysis tool, for capturing the profile of the workloads on the computer systems. Unlike most characterization tools, MICA is microarchitecture independent. With MICA, the profiles of the specific workload on the different hardware architecture are the same. However, the workloads are required to be compiled by the same compiler and run on the same operating system.

MICA applied a principal components analysis (PCA) in collaboration with a genetic algorithm (GA) in order to find the parameters, gathering by the binary instrumentation tools, that can represent the microarchitecture-independent characteristics of the workloads. From the total of 47 characteristics, only eight of them were obtained:

1. Probability of a register dependence distance ≤ 16
2. Branch predictability of per-address, global history table (PAg) prediction- by-partial-matching (PPM) predictor
3. Percentage of multiply instructions
4. Data stream working-set size at 32-byte block level
5. Probability of a local load stride = 0
6. Probability of a global load stride ≤ 8
7. Probability of a local store stride ≤ 8
8. Probability of a local store stride ≤ 4,096

The MICA project were experimented on 118 benchmarks from 6 benchmark suites. The results showed that the 8 microarchitecture-independent metrics could efficiently characterize the workloads. Moreover, MICA could provide more information and accuracy than the microarchitecture-dependent workload characterization tools.

In our work, MICA would be used for capturing the characteristic of the dwarfs and the benchmarks. The eight metrics would be adopted in the workload classification model training phase and the runtime equations construction phase. Moreover, they would also be used as the input of both workload classification model and the runtime estimation model.

2.2.3 Berkeley's Dwarfs and Benchmark suites

To select the appropriate runtime estimation model for the predicting the runtime of the application, the application has to be classified into a class. The classification results would be used to choose the runtime equation for the application. The classes of the application defined in our framework are referred from the Berkley's Dwarfs taxonomy [14].

In the year 2006, the researchers at the University of California at Berkeley discovered that the scientific applications generally have the similar computation patterns and data movements which should be categorized into a certain number of types. The Berkeley's Dwarfs, which are the classes of scientific applications, were then introduced. The similarity in computation behavior and data flow was used to define the membership in the class. However, there was no certain algorithmic calculations or numerical methods defined in each class since the applications with similar behaviors can be implemented differently. Currently, there are thirteen dwarfs [15]. Details of each dwarf are explained below.

Dense linear algebra (dense): the data is appropriate to be represented as the dense matrices/vectors. This kind of application generally has the unit-stride data access; meaning that the elements of array are read/written in sequence. Examples of dense linear algebra application are block tri-diagonal matrix and lower-upper symmetric Gauss-Seidel.

Sparse linear algebra (sparse): the data generally contains a large number of zero values. Therefore, the data is stored in the special format rather than the simple array in order to improve the efficiency of data access. The formats, for example; compressed sparse row matrix (CSR), compressed sparse column matrix (CSC), and dictionary of keys (DOK), create a list of the positions of the non-zero elements and store them in another array. The conjugate gradient application is an example of sparse linear algebra.

Spectral methods (spectral): the data for the spectral methods is transformed to the frequency domain from the time or spatial domain. The execution usually involves add-multiple operations and some specific data transformation pattern. Such computing pattern is called 'multiple butterfly stages'. The application that can be categorized in the spectral methods is Fourier transform (FFT).

N-body methods (nbody): the N-body methods compute the interactions between the data points in each time step. The n-body can be implemented in either particle-particle or hierarchical particle approach. The difference between the two approaches is interactions calculation. For particle-particle method, each data point depends on all other points, whereas, for the hierarchical particle method, each point depends on multiple points. The N-body simulation in astrophysics, which studies movement of bodies in the universe, is an example.

Structured grids (sgrid): the data points are stored in a simple grid and updated through time. In each time step, the value of each point is calculated by using the values of neighbors. One example of the structured grids is Multi-Grid, Scalar Penta- diagonal.

Unstructured grids (ugrid): unlike the structured grids, the unstructured grids store data points in the data structure, such as a linked list where the locations of data and the neighbor are tracked. The updates of all data points involve large indirect memory references since the locations of the points have to be looked up from the list before updating. Unstructured adaptive is an example of unstructured grid.

MapReduce (mapred): the model comprises of two execution phases – map and reduce. The map function performs the data filtering and sorting, while the reduce function aggregates the data. In the map phases, the tasks are independent, i.e., no communication is required between the working processes. In contrast, the global communication is important in the reduce phase. The example of MapReduce is a Monte Carlo application.

Combinational logic (clog): the combination logic applications perform bitwise operations on the data, which can achieve high computing throughput. This kind of applications repeatedly performs simple operations on a large amount of data. An example is cyclic redundancy check.

Graph traversal (grapht): data is represented in graphs. In order to perform a computation or search, the algorithm traverses through to a graph. Due to the structure of a graph, visiting nodes in a graph involves a great deal of random memory accesses. Examples of this include breadth-fist search and bitonic sort.

Dynamic programming (dprog): this approach solves a problem by dividing it into the smaller sub-problems. The sub-problems are solved in sequence from the smallest to the largest. The solution for the larger problem requires the answers of the smaller ones. The well-known problem, which the dynamic programming can be applied, is 0-1 knapsack.

Backtrack & branch-and-bound (bb): this approach is suitable for searching the optimal solution for the problem with massive search space. The branch-and-bound divides the search space into a smaller region and finds the solution candidates from the sub-regions. The A-start algorithm is a well-known branch-and-bound algorithm.

Graphical models (graphic): similar to the graph traversal approach, the data for the graphical models is represented in graph. Nodes and edges of the graph represent the problem's variables and the conditional dependencies, respectively. Examples are Bayesian networks and Hidden Markov Models.

Finite state machines (fsm): the behavior of the algorithm is defined in stages. The change of a current stage depends on input of a triggering event or a condition.

The release of the Berkeley's Dwarfs taxonomy results in the implementation of Dwarfs benchmark suites. According to the Berkeley's research [14], some of the benchmarks in NAS Parallel Benchmark (NPB) [16], from NASA, could be categorized as Dwarfs. NPB provides the parallel numerical aerodynamic simulation programs for testing the

performances of HPC platforms. NPB can only be run on multicore CPUs platform. In addition, the programs in NPB imitate the computation behavior and data movement in the computational fluid dynamics applications. For each application, the size of a problem, which is dependent to the input size, is divided into the predefined classes [17]: 1) S - small size, 2) W - legacy workstation size, 3) A, B, C - standard size (4 times bigger from one class to another), and D, E, F - large size (16 times bigger from one class to another).

Rodinia benchmark suite [18] was implemented by a group of researchers at University of Virginia. This benchmark suite consists of the diverse range of applications and kernels that cover six classes of Berkeley's Dwarfs [19]. The classification of dwarfs was based on the dwarf taxonomy given by Berkeley. The performance of the benchmark on the parallel platforms, which are multicore CPUs (OpenMP) and GPUs (CUDA), was measured.

In 2010, Berkeley released a Testbed for Optimization ResearCH or TORCH benchmark suite [20]. TORCH comprises 13 different classes of kernels written in C and MATLAB. Each class in the benchmark represents one of the Berkeley's Dwarfs. Similar to other benchmarks, TORCH were tested in various aspects including scalability, solution verification, and solution quality. Moreover, each kernel in the benchmark suite was mapped to the existing benchmarks in order to validate the practicality of TORCH.

These computing benchmarks will be run in the empirical study to collect the profile of the benchmarks. The profile collection will be used in a classification model training and runtime estimation equation construction. In order to avoid the redundant characteristic among classes, we used only seven classes of dwarfs – *dense*, *sparse*, *spectral*, *nbody*, *sgrid*, *mapred*, and *grapht*. The *ugrid*, *bb*, *graphic*, and *fsm* applications involve graph traversals in the calculation, therefore; using only *grapht* was sufficient in our framework. Moreover, the characteristics of *clog* and *dprog* can be represented by *nbody* because they repeatedly perform the operations to complete the task.

2.2.4 Optimization Algorithm

The runtime of the application is predicted by using the mathematical model that represent the relationship between the application's attributes (the set of MICA and Perf metrics) and the runtime. To build the mathematical model, we have employed the optimization method. The input of the optimization algorithm is a set of application attributes. The output is the runtime prediction equation.

Details on this section are the reviews of the optimization techniques. We mainly focus on the heuristic-based algorithms because of the efficiency in massive application scheduling. Heuristic-based scheduling algorithms can be categorized into two groups; heuristic algorithm and meta-heuristic algorithms [21]. Although these two categories apply a heuristic process in solution searching, their search methods are different. Both types of the heuristic based algorithm apply the exploration concept for searching the good-enough solution in an unexplored search space in order to avoid local optimum solution. For meta-heuristic, an exploitation concept is integrated in the algorithm for intensive finding a new solution. From the exploitation concept, meta-heuristic keeps

improving for the best solution found in previous searches until the search boundary, for example; iteration number is reached.

The heuristic [22], which is an experience-based solution searching method, does not search for the best solution, rather for a good enough solution. Therefore, not all possible solutions are to be considered. This kind of approach is suitable for the problem with a massive search space. Adopting a similar method, the meta-heuristic approach calibrates a candidate solution against a quality measurement, for example; fitness value, to derive a good-enough solution from a solution space. Artificial Bee Colony (ABC) [23] is the meta-heuristic scheduling algorithm that we chose because the performance of ABC, based on several researches [24],[25],[26], outperformed the other algorithms.

ABC mimics the food source searching behavior of bees. It consists of three phases; employed bee, onlooker bee, and scout bee phases.

The sequence of ABC algorithm can be shown in Figure 2.1.

Artificial Bee Colony Algorithm	
1	Randomly generate initial solutions
2	**WHILE** Termination criteria are not satisfied **DO**
3	Employed bees find better food sources in the adjacent area
4	Onlooker bees find better food sources around the existing sources based on the employed bee waggle dance
5	Scout bees search for the new food sources
6	Keep the best so-far food sources
7	**END WHILE**

Figure 2.1 ABC Algorithm

The ABC algorithm initially generates a set of feasible solutions, which are the food sources. In order to discover better food sources, three types of bees iteratively perform different tasks for developing the food sources. The employed bees are responsible for searching better food sources in the neighborhood. Then, the employed bees will perform the waggle dance to present the goodness of the discovered food sources. The onlooker bees will forage in the vicinity of existing food sources depending on the dance of employed bees. Thus, the best food sources have more probabilities to be visited by the onlooker bees. In contrast, the food sources that are arid will be dropped and replaced by new sources that have been found by the scout bees. The best food sources will be kept in each iteration until the stopping criterion is met.

The advantage of the ABC is it has the exploitation and exploration phases which enable thorough searching over the solution space. The ABC algorithm using a mathematical equation in solution adapting phase which is more efficient for continuous problems. However, the rounding technique can be applied in order to transform the solution to be discrete.

With this approach, appropriate mathematical models to predict the runtime of the applications are constructed. The detailed method of how to build the model by using ABC will be presented in the next chapter.

CHAPTER 3 RUNTIME ESTIMATION FRAMEWORK

As mentioned in the previous chapter, our proposed framework can be utilized to estimate the runtime of workloads with an unknown profile in a cloud computing environment. The unknown profile process can be called a 'black-box application'. The overall sequence of our method is shown in **Figure 3.1**.

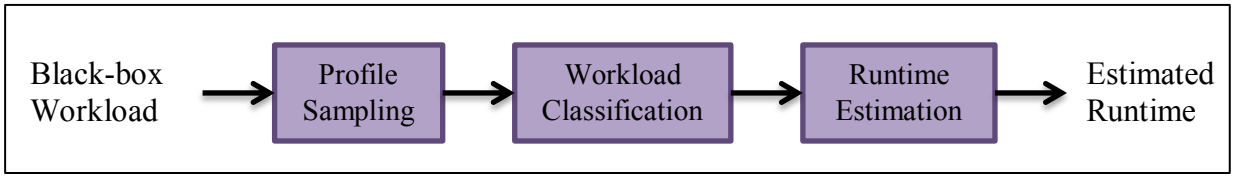

Figure 3.1 The Overall Methodology

As illustrated in **Figure 3.1**, there are three main steps in the proposed framework. Once a black-box application is submitted to the system, a profile of the workload will be collected. This step is called 'profile sampling'. In this step, the application will be run for a certain period for sampling its profile, which is a set of MICA metrics and Perf metrics.

Then, the workload's profile will be fed to a workload classification model in order to categorize the workload into a class. The class contains the workloads that have similar execution behaviors. As mentioned in the previous chapter, the classes of workloads are defined based on the taxonomy of Berkley's Dwarfs. We used only 7 out of 13 classes of dwarfs because some of them had repeating characteristics as others. Therefore, our framework and experiments are presented based on the 7 dwarfs. The 7 classes of dwarfs are shown in Table 3.1. Notice that the notations of dwarfs defined in Table 3.1 will be used throughout the report. The classification information will be used for selecting an appropriate runtime prediction model for the workload.

Table 3.1 The Class of Dwarfs Used in This Framework

Dwarf's 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*

In the last step, the runtime of the workload will be predicted by using the mathematical model selected from the previous step. The input attributes for the runtime estimation model are both MICA metrics and Perf metrics. In addition to our framework, the runtime can be used further in workload scheduling in case that a profile of the process is unknown.

The elaboration on each step in our methodology will be explained in three separate subsections: profile collecting and sampling, workload classification model, and runtime estimation model.

3.1 Workload Profiles Collecting

The profile of the application, which comprises 12 attributes, is used as input for both a workload classification model and a runtime estimation model. In the model construction phase, the profiles of the several benchmarks are also used as a training data. This section explains the workload profiling procedure.

In our work, we collected a total of 12 metrics from two workload characterization tools: MICA and Perf (a Linux profiling tool). Eight microarchitecture-independent metrics came from MICA, whereas the rests from Perf. However, the performance metrics, from Perf, are dependent to hardware architecture. Unlike a user name and a project name, these metrics can represent the actual execution characteristic of the applications. The Table 3.2 shows the list of metrics used in our framework. Notice that the notations of the metrics presented in Table 3.2 will be used throughout the rest of the report.

Table 3.2 The List of Metrics

Tool	Metric	Notation
MICA	1. Probability of a register dependence distance ≤ 16	*reg_age_cnt_16*
	2. Branch predictability of per-address, global history table (PAg) prediction-by-partial-matching (PPM) predictor	*PAg_mispred*
	3. Percentage of multiply instructions	*arith_cnt*
	4. Data stream working-set size at 32-byte block level	*data_stream*
	5. Probability of a local load stride = 0	*mem_read_local_stride_8*
	6. Probability of a global load stride ≤ 8	*mem_read_global_stride_8*
	7. Probability of a local store stride ≤ 8	*mem_write_local_stride_8*
	8. Probability of a local store stride ≤ 4,096	*mem_write_local_stride_4096*
Perf	9. CPU clock	*cpu*
	10. Task clock	*task*
	11. Page faults	*fault*
	12. Context switches	*cs*

In fact, the Perf tool can measure both hardware events (e.g., CPU cycles, instructions, cache references) and software events (e.g., CPU clock, task clock, page faults). However, we could collect only the software event metrics because the virtual machines in the cloud do not allow users to access the hardware events. Therefore, only 4 software events were used.

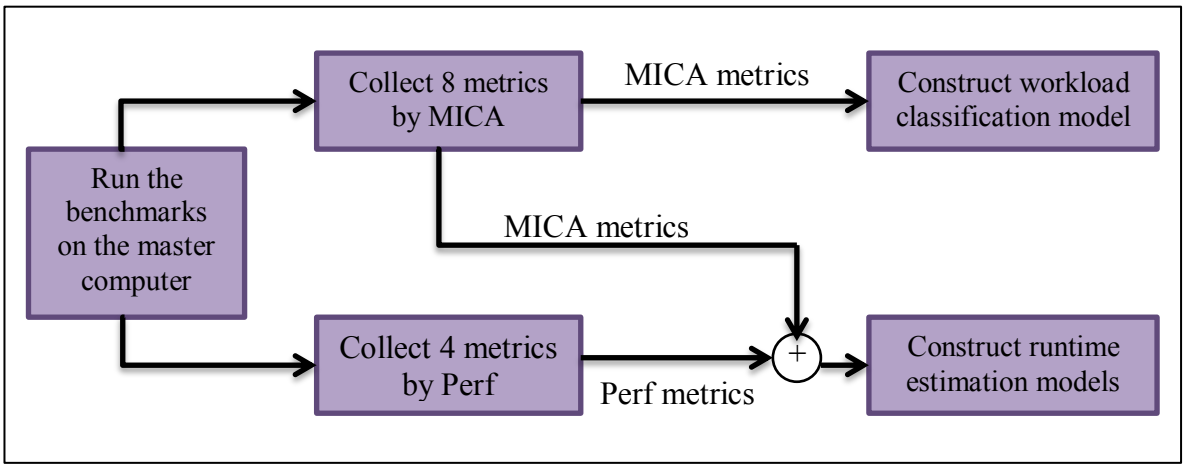

Figure 3.2 Benchmark Profiles Collecting Workflow

The sequence of benchmark profiles collecting is illustrated in **Figure 3.2**. We performed this process to collect the training data for model construction. In order to collect the profiles, the benchmarks were run on the master computer and the metrics are captured by MICA and Perf. Once the set of metrics are obtained, it will be used in the models construction phase. This step requires a large amount of data, which is a set of workload profiles, to train the models. The workload classification model needs the profile to build a decision tree for categorizing the workloads into classes. The runtime estimation models apply the linear regression technique to fit the attributes of the profiles to the mathematical models. More details on the models construction procedures will be presented in this chapter.

As shown in **Figure 3.2**, only MICA metrics are used in workload classification model training because the classification results should be independent to the hardware architectures. Thus, MICA metrics, which are microarchitecture-independent metrics, are employed. However, all the collected metrics will be used in runtime estimation model training since the runtime of a workload must be specific to the machine. Therefore, Perf metrics, which are dependent to machines, are used in collaboration with MICA metrics in the runtime estimation models construction.

The master computer used for collecting the profile of the workloads runs the Ubuntu 12.04 operating system. The compilers for C and C++ are gcc-4.4 and g++-4.4, respectively. Since MICA is the extension of a pin tool, we had to install the pin tool version 2.11 on the computer. Notice that the later versions were not compatible with MICA. We installed the latest version of MICA, version 0.40, on our machine. The Perf tool that we installed was version 3.2.53.

The training data that we used in models construction step are the profiles of 20 benchmarks, from 3 benchmark suites. The classes of the benchmarks were known. The period of profile collecting was equal to the execution time of each benchmark. We also adjusted the input parameters in order to obtain the profiles of the workloads with various input sizes and runtimes. The list of benchmarks is shown in Table 3.3.

Table 3.3 The List of Benchmarks

Benchmark		Benchmark Suite
kmeans	hotspot	Rodinia
lud	lavaMD	
nn	leukocyte	
heartwall	particle	
lu (A,B,S)	sp (A)	NPB
ep (A,B,C,S)	cg (A,B,C,S)	
dense	monteCarlo	TORCH
integerSort	nbody2d	
quickSort	sparse	
radixSort	spectral	

In our work, the MICA and performance metrics (Perf metrics) were collected separately because machine did not allow multiple tools to analyze the workload at the same time. To collect the profile of the workload, the Linux shell scripts were used to leash the profile gathering process.

For MICA metrics, we captured the data every one million instructions. Then, we obtained the set of raw metrics in a file. The raw metrics were calculated using the formulas given in a manual in order to retrieve the MICA metrics. At the final step, the values in each MICA metric were averaged. Therefore, the MICA metrics that we used throughout the project were the averaged values of a full-length run of the workload.

In order to measure the performance metrics, a simple 'perf' command was run. Unlike MICA, the metrics-collecting interval cannot be defined. Thus, we executed the 'perf' command every 2 seconds instead. The performance metrics were contained in a file. As same as the MICA metrics, the mean of the values in each performance metric were computed.

In the experiment, the method to sample the profile of the workload is the same as the procedure that is presented in this section. However, the sampling period was shorter. Details of profile sampling methods will be presented in the next chapter.

3.2 Workload Classification Model

The classification model is used for categorizing an unknown-profile workload into a class of dwarfs. The classification result is further utilized in the runtime prediction step. As mentioned earlier, our proposed framework was based on the 7 Berkley's dwarfs. We can map the benchmarks to the Berkeley's dwarfs as shown in Table 3.4.

Table 3.4 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
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

To train the model, we collected 255 different workload profiles as a training set. Only MICA metrics were used in this step because, as mentioned in the previous section, we needed the hardware-independent classification results. The attributes of the model were the MICA metrics, which are the floating number between 0 and 1. The labels were the 7 classes of dwarfs: *dense*, *sparse*, *spectral*, *nbody*, *sgrid*, *mapred*, and *grapht*.

We plotted the Kiviat diagram for each dwarf, shown in Figure 3.3, in order to analyze the similarity of the different applications in the same class of dwarfs. Each axis represents the MICA metrics. The polygon in each diagram represents the plot of each application in the specific class of dwarfs (see the list of applications in the same class in Table 3.4). The values plotted in the graph are normalized in Z-scores, equation (3.1).

$$Z = \frac{X - \bar{x}}{sd} \tag{3.1}$$

where Z is Z-score
X is the value to be normalized
$\bar{x}$ is the mean of all values
sd is the standard deviation of all values

From Figure 3.3, the benchmarks in the same class have similar MICA metrics. This can be seen from the shapes of the polygons in the same graph. Therefore, we could use this data in model training because the applications in the same class could establish similar program characteristics.

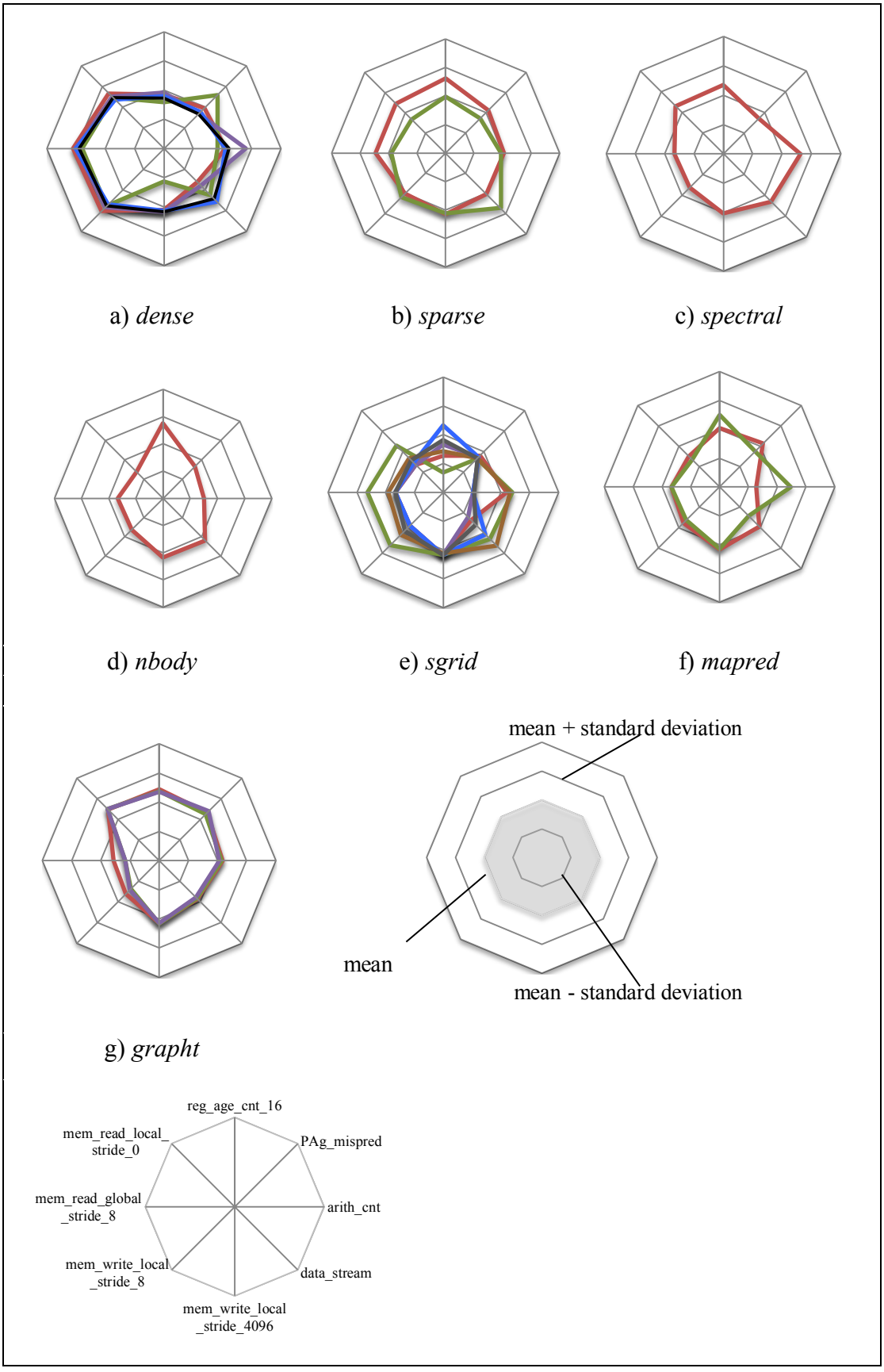

Figure 3.3 Kiviat Diagrams of MICA Metrics

3.2.1 Workload Classification Model Construction

In order to select the classification method, we used Weka, which is a data mining analysis tool, to run several classification methods, Bayesian Network, k-NN, Rule-Based, and Decision Tree, to compare the classification results. It turned out that every approach could give comparable good results. We then selected a C4.5 algorithm to build a classification model, which is a decision tree, since a decision tree is simple, and the result is easy to be interpreted. Moreover, C4.5 has been used widely in the real applications [27],[28].

The results from the decision tree algorithm are a set of rules, derived from the decision tree, for classifying the application into classes. The rules are represented in the Boolean expressions, which ^ and ∨ denote *and* and *or* operations. The conditional expressions use <, ≤, >, and ≥ to represent *less than*, *less than or equal*, *greater than*, and *greater than or equal* operations, respectively. Moreover, we used the following notation to represent the name of MICA metrics:

- A denotes *reg_age_cnt_16*
- B denotes *PAg_mispred*
- C denotes *arith_cnt*
- D denotes *data_stream*
- E denotes *mem_write_local_stride_4096*
- F denotes *mem_write_local_stride_8*
- G denotes *mem_write_global_stride_8*
- H denotes *mem_read_loca_stride_8*

Since there was a rule for every class, there were 7 rules in total. To use the rules to classify the application, the MICA metrics must be known. The application would be categorized into the class only if all the conditions in the rule of that class were valid. The rules are as follows:

Rules for *dense*

(A ≤ 0.658527 ^ F ≤ 0.001908 ^ B ≤ 0.00019 ^ G ≤ 0.000094) ∨
(A ≤ 0.648527 ^ F ≤ 0.001908 ^ B > 0.00019 ^ H > 0.000014 ^ F ≤ 0.000027) ∨
(A ≤ 0.648527 ^ F > 0.001908) ∨
(A > 0.64852 ^ H > 0.000014 ^ B ≤ 0.004874 ^ E > 0.99348)

Rule for *sparse*

(A > 0.648527 ^ H ≤ 0.000014 ^ E ≤ 0.99072) ∨
(A > 0.648527 ^ H > 0.000014 ^ B ≤ 0.004874 ^ E ≤ 0.99348 ^ B > 0.001732) ∨
(A > 0.648527 ^ H > 0.000014 ^ B > 0.004874 ^ D ≤ 3833.065186 ^ F > 0.000011) ∨
(A > 0.648527 ^ H > 0.000014 ^ B > 0.004874 ^ D > 3833.065186)

Rule for *spectral*

(A > 0.648527 ^ H > 0.000014 ^ B ≤ 0.004874 ^ E ≤ 0.99348 ^ B ≤ 0.001732)

Rule for *nbody*

(A > 0.648527 ^ H ≤ 0.000014 ^ E > 0.99072 ^ B ≤ 0.000109 ^ E > 0.999893)

Rule for *sgrid*

(A ≤ 0.648527 ^ F ≤ 0.001908 ^ B ≤ 0.00019 ^ G > 0.000094) ∨
(A ≤ 0.648527 ^ F ≤ 0.001908 ^ B > 0.00019 ^ H ≤ 0.000014) ∨
(A ≤ 0.648527 ^ F ≤ 0.001908 ^ B > 0.00019 ^ H > 0.000014 ^ F > 0.000027) ∨
(A > 0.648527 ^ H ≤ 0.000014 ^ E > 0.99072 ^ B > 0.000109 ^ B ≤ 0.006144 ^ E > 0.996586)

Rule for *mapred*

(A > 0.648527 ^ H ≤ 0.000014 ^ E > 0.99072 ^ B ≤ 0.000109 ^ E ≤ 0.999893) ∨
(A > 0.648527 ^ H ≤ 0.000014 ^ E > 0.99072 ^ B > 0.000109 ^ B ≤ 0.006144 ^ E ≤ 0.996586) ∨
(A > 0.648527 ^ H ≤ 0.000014 ^ E > 0.99072 ^ B > 0.000109 ^ B > 0.006144)

Rule for *grapht*

(A > 0.648527 ^ H > 0.000014 ^ B > 0.004874 ^ D ≤ 3833.065186 ^ F ≤ 0.000011)

The stratified 10-fold cross-validation was applied to the model in order to measure the quality of the decision tree. The stratified cross-validation ensures that the testing data in each fold is sampled from all classes. Our decision tree yielded a high accuracy of 96.89 percent. The experiment on the classification model will be presented in Chapter 4.

3.3 Runtime Estimation Model

Our work aimed to predict the runtime of the workloads on the cloud. Thus, three types of Amazon EC2's on-demand instances were adopted. Based on the survey, we selected the instances that were likely to be chosen for a high performance computing purpose. The list of virtual machines used in this framework is shown in Table 3.5.

Because there were 7 dwarfs with different profiles, each instance type required 7 runtime estimation models. Each model could only estimate the runtime for a specific dwarf on a specific machine. For this reason, a benchmark or a workload had to be accurately classified into a dwarf on specific machine before predicting a runtime. Therefore, there was 21 runtime prediction models in total.

The runtime prediction model is a mathematical equation that describes the relationship between the metrics, an input size, and a runtime. The runtime is a dependent variable, but the rest are not.

In the previous section, we described the procedure to obtain a set of metrics from MICA and Perf. However, there is another important input to the prediction model, which is an input size of the workload. The input size has to be well defined since it can affect to a precision of a prediction. We then defined the way to normalize the input size for each dwarf.

Table 3.5 Instances of Amazon Web Service that Used in the Framework

Name	Type	Capacity
m1.large	General purpose	4 ECUs[1] 2 vCPUs[2] 7.5 GB Memory 2 x 420 GB Storage Moderate network performance
c1.xlarge	Compute optimized	20 ECUs[1] 8 vCPUs[2] 7 GB Memory 4 x 420 GB Storage High network performance
m2.2xlarge	Memory optimized	13 ECUs[1] 4 vCPUs[2] 34.2 GB Memory 1 x 850 GB Storage Moderate network performance

[1]ECU is a computing unit of Amazon EC2, which is equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processer
[2]vCPU is a number of virtual CPU

In the following subsections, the normalization of the input sizes is presented in the first section, and the detailed explanation on the steps to derive a mathematical equation for each dwarf is given in the last section.

3.3.1 Input Size Generalization

In addition to the 12 metrics, the input size of the workload is also required in the runtime prediction step. Basically, the different applications may have different ways to define the input size, even for the applications in the same class of dwarfs. We defined the method to normalize the input size for each type of dwarf. The details are provided below.

dense: a data of *dense* is usually represented in a matrix/vector. The algorithm generally iterates over the matrix/vector in order to access, read or write, the elements. Thus, a dimension of the matrix/vector importantly affects the runtime of the *dense* application. We defined the input size of *dense* as follows:

$$(input\ size)_{dense} = n \times m \tag{3.2}$$

where n is the number of rows of a matrix/vector
m is the number of columns of a matrix/vector

sparse: a data of *sparse* is similar to *dense*. However, it contains a large number of zero elements. Therefore, the data is stored in a special matrix/vector, which memorizes only the non-zero elements, in order to optimize the memory space. We then defined the input of *sparse* as follows:

$$(input\ size)_{sparse} = nnz \tag{3.3}$$

where nnz is the number of non-zero elements

spectral: the *spectral* methods transforms the data from a spatial domain to a frequency domain. The following equation can be used for normalizing an input size of the two dimensional data.

$$(input\ size)_{spectral} = n \tag{3.4}$$

where n is the number of data to be transformed

nbody: the *nbody* applications calculate the interactions between the data points. The data points are computed through the defined time steps. Thus, the input size is directly respected to the number of particles and the time steps. The input size normalization for *nbody* is shown as follows:

$$(input\ size)_{nbody} = n \times (time\ steps) \tag{3.5}$$

where n is the number of particles/bodies
time steps is the number of time steps to be simulated

sgrid: for *sgrid*, data is stored in a grid and updated in every time step by exploiting the values of its neighborhood. The input size equation is as follows:

$$(input\ size)_{sgrid} = n \times m \times (time\ steps) \tag{3.6}$$

where n is the number of rows
m is the number of columns
time steps is the number of time steps to be computed

mapred: the *mapred* applications usually involve a large amount of data. The input size then depends on the number of data as shown below:

$$(input\ size)_{mapred} = n \tag{3.7}$$

where n is the number of data item

grapht: for *grapht*, the number of nodes in a graph strongly affects a value of input size. Thus, the input size can be computed as follows:

$$(input\ size)_{grapht} = n \tag{3.8}$$

where n is the number of nodes in a graph

Notice that this step requires an expert to determine the input size of the workload. The input size for each dwarf is a simple product of the input parameters. However, the calculation of the input size does not need to be complex because, in the next step, it will be composed in the runtime prediction model.

3.3.2 Runtime Estimation Model Construction

From the previous steps, the 12 metrics (from MICA and Perf) and the input size were obtained. In order to construct the runtime estimation model for each dwarf, the relationship between the inputs and a runtime has to be defined. Therefore, we collected a training data, which contained various input parameters and runtimes. Each data record consisted of the 12 metrics, an input size, and a runtime of the workload. Due to some availability of the benchmark functions, we could not collect the data for the dwarfs evenly. The minimum and maximum number of data records were 34 and 50, respectively.

In order to determine such relationship, we adopted the Artificial Bee Colony algorithm, also known as ABC. The ABC algorithm mimics a foraging behavior of bees. In our context, the solutions were the food sources of bees. The goal of ABC is to find the mathematical model that can best describe the relationship between the inputs and a runtime, in linear time. The brief steps of ABC are shown in **Figure 3.4**.

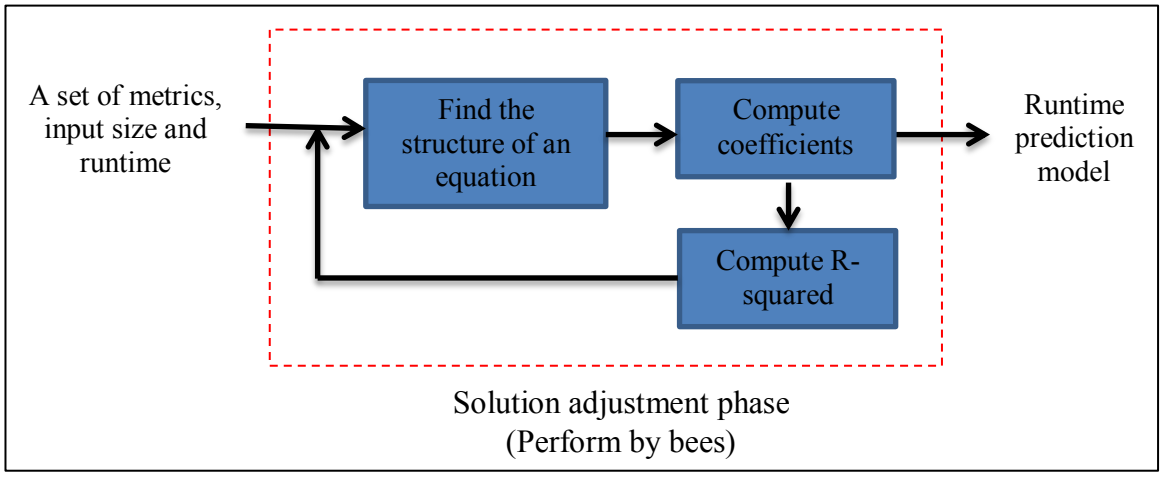

Figure 3.4 ABC Steps

From **Figure 3.4**, ABC starts by finding the possible structure of the equation. For example: $runtime = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_0$; where β_i and x_i are coefficients and independent variables. The coefficients of the equation are not known in the first place, therefore; ABC computes the coefficients by using linear regression in the next step. Before applying linear regression to the training data, we ensured that the data were normally distributed by plotting probability graphs of the runtimes. The graphs indicated that the runtimes of all data sets were normally distributed. Under normality assumption, linear regression can be efficiently used to fit data to the model. To evaluate an accuracy of the model, the R-squared is then calculated. The more R-squared is closer to 1 (100%), the higher the accuracy of the prediction model. The R-squared can be computed by using equation (3.9).

$$\text{R-squared} = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{3.9}$$

where y_i is an actual value
f_i is a predicted value
$\bar{y}$ is a mean of the actual values

After that, the structure of the equation is iteratively developed. Once ABC stops, the runtime prediction model is derived.

The objective of ABC is to minimize the R-squared value of the equation. Thus, the objective function and fitness value of the ABC algorithm are shown in equation (3.10).

$$\textbf{Maximize} \quad \text{fitness} = \text{R-squared}(\text{solution}_i) \quad ; 1 < i < n \tag{3.10}$$

where n is a total number of bees

In this work, interactions between applications' attributes were not considered because examining variables interactions would significantly increase a size of the search space, and a good enough solution might not be obtained in the linear time. For ABC, the solution is encoded in 3 main arrays: *Term*, *Function*, and *Operation*. Details for each array are presented in Table 3.6.

Table 3.6 Arrays for ABC

Array	Description	Possible Values	Length	
			Max	Min
Term (*T*)	It stores the selected input parameters of the workload.	cpu_clock (*A*) task_clock (*B*) page_fault (*C*) context_switch (*D*) arith_cnt (*E*) PAg_mispred (*F*) reg_age_cnt_16 (*G*) data_stream (*H*) mem_read_local_stride_8 (*I*) mem_read_global_stride_8 (*J*) mem_write_local_stride_8 (*K*) mem_write_local_stride_4096 (*L*) input_size (*M*)	11	2
Function (*F*)	It stores the selected functions for the input parameters (*p*).	p = p log(p) = $\log_{10}(p)$ ln(p) = $\ln(p)$ power(p,m) = p^m , $2 \leq m \leq 4$ sqrt(p) = $\sqrt{p}$	11	2
Operation (*O*)	It stores the mathematical operations.	add (+) subtract (-)	10	1

The structure of the solution is shown in Figure 3.5.

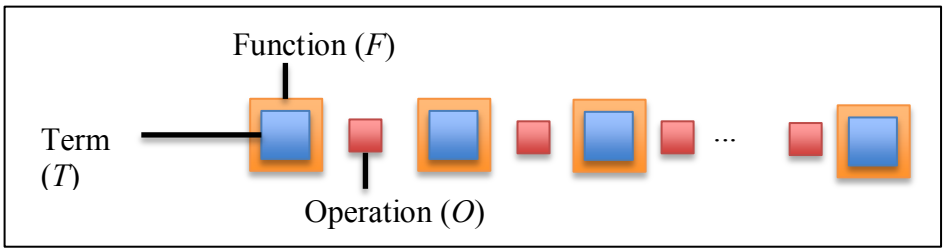

Figure 3.5 Structure of ABC's Solution

One solution must consist of at least two terms, two functions, and one operation. Moreover, we limited a number of terms to not exceed 11 in order to control a number of feasible solutions. For this problem, a size of search space became $13^{11} \times 5^{11} \times 2^{10}$, about 6.9×10^{21}.

There were 3 types of bees responsible for finding the best food source; employed bee, onlooker bee, and scout bee. At the beginning of the algorithm, each employed bee randomly created the solution. Thus, the number of initial solutions was equal to the number of employed bee in the colony. Then, the solutions were iteratively developed until the termination criterion was satisfied. In this project, the algorithm would stop when a number of calculation-iterations reached a defined threshold.

1. Employed bee phase
   For each loop, each employed bee improved its solution. A new solution was developed by randomly selecting the positions in the arrays to adjust. For each selected position, a probabilistic value, a new random value, and a neighbor's value were used in order to compute the new value. The formulas are shown in equation (3.11) to equation (3.14).

$$index_A = \textbf{Random}(\text{Integer} < length_A) \quad (3.11)$$
$$\text{oldValue} = A\,[\text{index}_A] \quad (3.12)$$
$$newVal = \textbf{Random}(\text{Integer} \in possible_A) \quad (3.13)$$
$$A[index_A] = oldVal + \textbf{Random}(-1,1) \times (oldVal - newVal) \quad (3.14)$$

where $index_A$ is a selected index of the array A to adjust
$length_A$ is the number of items in the array A
$possible_A$ is a set of possible values in the array A
$oldVal$ is an old value in the array A at $Index_A$
$newVal$ is a new value that is randomly selected

2. Onlooker bee phase
   In each round, an onlooker bee selected a solution from an employed bee by using probabilistic selection shown in equation (3.15).

$$P_i = \frac{\boldsymbol{Fitness}(Solution_i)}{\sum_{i=1}^{n}(Solution_i)} \quad ; 1 < i < n \quad (3.15)$$

   where n is the total number of bees

   Then, it adjusted the solution by using the same method as employed bees.

3. Scout bee phase
   After the selected solution was adjusted in the earlier phases, the fitness value of adjusted solution was then calculated and compared to the fitness value of the original solution. If the adjusted solution got better, it would replace the original one. On the other hand, if it was poorer, the original solution would not be replaced, however, the un-improvement counter of this solution would be increased by one.

   In this phase, if the value of un-improvement counter reached the threshold, this solution would be dropped and it would be replaced by a new solution, which was randomly created.

Due to a large search space, we adopted the parallel computing in order to improve the running performance of ABC. The algorithm was run on the 12-core computers with 32 GB-memory. The number of bees in a colony was 3600 in total (1200 for each type of bees). The un-improvement and termination thresholds were set to 10 and 10000, respectively. When the number of iterations reaches the termination threshold, the algorithm will stop and the best-so-far runtime prediction equation will be obtained. In other words, the algorithm runs until the solution no longer improve for 10 iterations or when the iteration of 10,000 is reached.

Because ABC applies a heuristic method to search for a good enough solution in limited time, the best solutions from ABC may not be the same every time even for the same training data. In our work, we ran ABC 5 times on each data set and selected the runtime equation with the highest R-squared.

The runtime prediction models and R-squared values for each dwarf are presented below. There were three equations for each dwarf class for predicting the runtime of the application on the different machines in Amazon EC2. The obtained runtime is in second. The machines were General purpose, Compute optimized, and Memory optimized instance (see Table 3.5 for more details). The notations used in the equations are shown in Table 3.6 (the Possible Values column).

dense

- General purpose machine
  R-squared: 99.6%

$$runtime = 1.97 \times 10^8 + 0.508\ \mathrm{sqrt}(M) + 96277331\ G - 3020\ \ln(F) + 20268\ E + 2923\ \ln(J) - 434\ \mathrm{sqrt}(L) + 1.13\ L + 9457\ \ln(L) + 136290\ \mathrm{sqrt}(F) - 2.93\mathrm{E}{+}08\ \mathrm{sqrt}(G) + 55353417\ \ln(G) \quad (3.15)$$

- Compute optimized machine
  R-squared: 99.4%

$$runtime = -42746086 + 2500\ \ln(J) + 0.198\ L + 0.435\ \mathrm{sqrt}(M) + 9974\ \mathrm{sqrt}(H) - 25.7\ \mathrm{sqrt}(L) - 86\ \ln(D) - 12876458\ \ln(G) - 64697\ \mathrm{sqrt}(J) - 17687534\ G + 2.21\ \mathrm{sqrt}(C) + 60421185\ \mathrm{sqrt}(G) \quad (3.16)$$

- Memory optimized machine
  R-squared: 99.4%

$$runtime = 1.30 \times 10^8 + 0.312\ \mathrm{sqrt}(M) + 62857250\ G - 1365\ \ln(F) + 10739\ E + 1604\ \ln(J) - 191\ \mathrm{sqrt}(L) + 0.450\ L + 4805\ \ln(L) + 70132\ \mathrm{sqrt}(F) - 1.92 \times 10^8\ \mathrm{sqrt}(G) + 36668888\ \ln(G) \quad (3.17)$$

sparse

- General purpose machine
  R-squared: 99.8%

$$runtime = -27 - 17918\ \mathrm{power}(E,3) + 165\ \ln(M) - 14458\ \log(E) - 119\ \ln(D) + 29.9\ \ln(J) + 188741\ F + 28.2\ \mathrm{sqrt}(D) + 19914\ \mathrm{sqrt}(H) - 7706\ \ln(J) - 3692\ \mathrm{power}(G,3) - 220419\ H \quad (3.18)$$

- Compute optimized machine
  R-squared: 99.8%

$$runtime = -73.0 - 268\ \ln(M) - 45881\ \mathrm{sqrt}(I) - 161025\ G + 3.35\ \mathrm{sqrt}(M) + 244232\ K + 3107\ F + 2.56\ \ln(H) - 87542\ \mathrm{power}(K,3) + 118456\ \ln(G) - 312\ \log(F) - 272\ \mathrm{power}(E,3) \quad (3.19)$$

- Memory optimized machine
  R-squared: 99.8%

$$runtime = 18.6 + 2.93\ \mathrm{sqrt}(M) - 94.1\ \ln(M) + 42939\ \mathrm{sqrt}(I) - 11629\ H - 44.8\ \ln(L) + 100\ \ln(A) - 207\ B + 206\ A + 0.271\ \log(J) + 528\ \mathrm{sqrt}(K) + 9.81\ \log(I) \quad (3.19)$$

spectral

- General purpose machine
  R-squared: 95.9%

$$runtime = 1824 + 0.0389\ \mathrm{sqrt}(M) + 1\ \mathrm{sqrt}(A) - 574\ \log(L) - 74\ \ln(A) + 0.0288\ L - 0.02\ A - 3288\ \ln(G) - 42.0\ \ln(D) - 14\ \log(H) - 78252160\ \mathrm{I} - 10.7\ \ln(M) \quad (3.20)$$

- Compute optimized machine
  R-squared: 96.6%

$$runtime = 6793 + 0.0251\ \mathrm{sqrt}(M) + 53\ \mathrm{sqrt}(A) + 25755716\ I - 355\ \ln(L) - 9.21\ \ln(M) + 2381152\ \ln(B) - 2677\ G - 2381678\ \ln(A) - 0.31\ A + 7.28\ \mathrm{sqrt}(L) + 21.8\ \log(D) \quad (3.21)$$

- Memory optimized machine
  R-squared: 85.4%

$$runtime = -14439 + 1.05E{+}08\ I - 53690\ \ln(G) + 20.6\ \ln(D) + 3.95\ \ln(M) + 10.1\ \text{sqrt}(A) + 60236\ K + 67571\ G - 112010\ \text{sqrt}(K) + 0.000044\ C - 14.1\ \ln(L) - 17.9\ \ln(C) \quad (3.22)$$

nbody

- General purpose machine
  R-squared: 96.6%

$$runtime = -46 + 0.000090\ M + 0.000000\ \text{power}(C,2) - 2.4\ \text{power}(A,2) - 0.0144\ C + 2.4\ \text{power}(B,2) + 0.0564\ \text{power}(D,2) - 0.000002\ \text{power}(L,2) + 0.138\ \text{sqrt}(M) + 0.115\ L \quad (3.23)$$

- Compute optimized machine
  R-squared: 94.3%

$$runtime = -54 + 0.000085\ M + 0.00172\ \text{power}(A,3) + 0.00601\ C - 0.307\ A + 0.0757\ L - 0.00172\ \text{power}(B,3) + 0.000282\ \text{power}(D,3) \quad (3.24)$$

- Memory optimized machine
  R-squared: 82.6%

$$runtime = -8279 + 1.71\ \text{sqrt}(M) + 16812\ E - 1736\ \ln(M) + 3086400\ \ln(K) + 7.84\ \text{sqrt}(C) - 7403\ \ln(F) - 738107\ \text{sqrt}(H) - 70443\ \text{sqrt}(G) - 0.809\ L + 1331254\ \text{sqrt}(F) + 318\ \ln(L) \quad (3.25)$$

sgrid

- General purpose machine
  R-squared: 97.8%

$$runtime = 30774 + 139\ \text{sqrt}(M) + 1057953\ \ln(A) + 11395232\ B - 245502\ \ln(E) - 6030218\ \text{sqrt}(K) + 104301\ \ln(L) - 11396367\ A + 16.9\ L - 35676\ \text{sqrt}(D) - 7951\ \text{sqrt}(L) - 4755113\ \text{sqrt}(I) \quad (3.26)$$

- Compute optimized machine
  R-squared: 99.1%

$$runtime = 158662 + 29.5\ \text{sqrt}(M) - 262379\ \ln(A) + 120580\ \text{sqrt}(A) + 4735\ \ln(M) - 1725995\ \text{sqrt}(J) + 53564046\ I - 25.8\ \text{sqrt}(C) - 418\ \ln(H) + 234221\ F - 60054\ \ln(D) - 1651\ B \quad (3.27)$$

- Memory optimized machine
  R-squared: 97.6%

$$runtime = -354821 + 38.0\ \mathrm{sqrt}(M) - 3035229\ A - 740690\ K - 8732\ \mathrm{sqrt}(D) - 436570\ \mathrm{sqrt}(J) + 189079\ \ln(A) + 3035031\ B - 64082\ \ln(E) + 4.05\ L - 1913\ \mathrm{sqrt}(L) + 56561\ \log(L) \quad (3.28)$$

mapred

- General purpose machine
  R-squared: 99.7%

$$runtime = 306958 + 0.219\ M - 230720\ K - 15491\ \ln(D) + 67.7\ \mathrm{sqrt}(L) - 36\ \mathrm{sqrt}(C) - 3244\ \ln(L) - 740\ \log(C) + 3.30\ C - 1078\ \log(M) + 15.4\ B \quad (3.29)$$

- Compute optimized machine
  R-squared: 99.1%

$$runtime = -580 - 207\ \mathrm{sqrt}(M) + 0.00316\ \mathrm{power}(A,2) + 7.63\ C + 1761\ \ln(M) - 108\ D - 197\ \mathrm{sqrt}(C) + 1.11\ M - 0.000802\ \mathrm{power}(C,2) + 702\ \log(L) + 204\ \log(F) + 19.5\ \ln(J) \quad (3.30)$$

- Memory optimized machine
  R-squared: 99.8%

$$runtime = 338 - 23.1\ \mathrm{sqrt}(M) + 0.0253\ C + 0.174\ M - 0.0444\ L + 20.1\ \mathrm{sqrt}(L) + 0.175\ A - 1746\ \log(G) + 93.0\ \ln(M) - 542\ \log(L) - 37.4\ \log(I) + 35.1\ \log(J) \quad (3.31)$$

grapht

- General purpose machine
  R-squared: 92.4%

$$runtime = 1395 - 0.000075\ C + 0.203\ \mathrm{sqrt}(C) + 211\ \mathrm{sqrt}(F) - 84.1\ \mathrm{sqrt}(B) + 1.07\ B + 41\ G - 89\ \log(H) - 0.85\ \mathrm{sqrt}(L) + 0.00000043\ M \quad (3.32)$$

- Compute optimized machine
  R-squared: 91.6%

$$runtime = 993 + 0.000000\ M - 0.32\ \mathrm{sqrt}(A) - 747\ F - 456\ K + 0.155\ \mathrm{sqrt}(C) - 24\ \log(F) - 3251\ H - 681\ \mathrm{sqrt}(E) - 0.000050\ C \quad (3.33)$$

- Memory optimized machine
  R-squared: 94.8%

$$runtime = 11010 - 0.00675\ \mathrm{sqrt}(M) + 12744\ \log(K) - 1033333\ I - 0.0020\ L - 11029\ \mathrm{sqrt}(K) + 0.96\ \ln(M) - 0.000004\ C + 0.130\ \mathrm{sqrt}(C) + 0.00000038\ M + 99\ \mathrm{sqrt}(F) \quad (3.34)$$

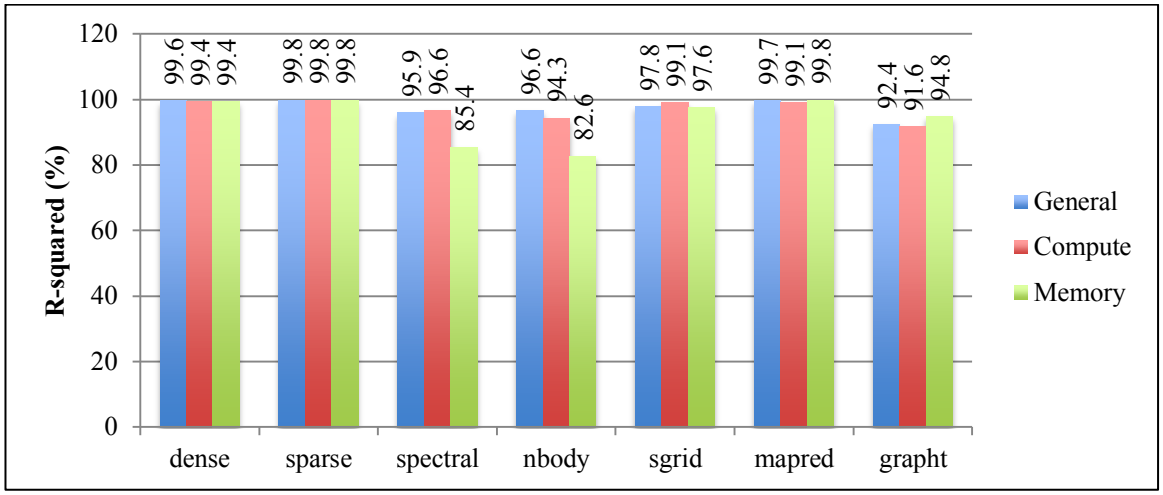

Figure 3.6 Percentage of R-squared Values of Dwarfs on Virtual Machines

The runtime estimation equations that obtained from ABC have high R-squared values as shown in Figure 3.6. R-squared values of almost all equations are higher than 90%. The runtime estimation equations obtained from ABC have high R-squared values as shown in Figure 3.6. R-squared values of almost all equations are higher than 90%. This implies that ABC could efficiently find the model that could describe the relationship between the inputs and a runtime of the workload in linear time.

In addition to methods mentioned in this section, the obtained runtime equations from ABC can be further improved by determining the correlations between variables and runtimes. Since ABC is a heuristic approach, some attributes might not be significantly correlated to the runtimes. Thus, removing the irrelevant attributes may improve the qualities of the models, and it would ensure that all attributes in the equations were significantly correlated to the runtimes. The simplest way to remove the uncorrelated attributes from the model is to try removing the attributes one by one and then recalculate R-squared. If new R-squared is better than the old one, the removed attribute might be insignificant to the model. Another approach is to calculate correlation measures between attributes and runtimes, and remove the attributes that are deemed as unimportant.

Furthermore, if new data records were added to the data set, ABC could compute a new model by adopting an original equation as the best so far solution and a seed solution. This method can significantly improve the performance of ABC since it does not have to start from the scratch. The experiments on the runtime prediction equations will be provided in chapter 4.

Although all runtime estimation models can achieve very high R-squared values, there is a limitation in our models. Our models might not be efficient to predict runtimes of applications that fit behaviors of multiple dwarfs because the applications that are used to train the models are single-patterned applications. To solve this problem, dwarf classes with mixed behaviors have to be added. For instance, workload classes should include the classes that represent the combinations of existing dwarf classes (for example, *dense* & *sparse* class, *dense* & *grapht* class, and so on).

In this section, we have presented the methodology to estimate the runtimes of the applications. Our framework is divided into 3 parts: profile colleting, workload

classification, and runtime prediction. The profile collecting used MICA and Perf to record the profile of the process. The profile, which is a set of attributes that represent the characteristic of the application, is an input of the classification and prediction models. After the profile collecting process was done, the profile was entered into the workload classification model to identify the class of the application. The partition of classes was based on the Berkley's Dwarfs classification. Once the class was known, the runtime prediction model for the application on the specific computer could be selected. The profile of the application was also used as the input of the runtime estimation model. The outcome from the model is a predicted runtime of the application.

CHAPTER 4 EXPERIMENTS AND RESULTS

This section gives the details on the experiments and results of our proposed framework. The experiments were divided into two parts. The difference between the two experiments was the benchmarks used in testing. One experiment adopted trained benchmarks, while the other used untrained benchmarks.

The experimental procedure for the two experiments followed the methodology that was presented in chapter 3. At the beginning, the profile of a benchmark was sampled by running it on the master computer for a short period (see the details of the master computer in section 3.1). The profile was collected by MICA and Perf tool. Profile collecting took only a minute because it was the lowest runtime in the training data. In the next step, the runtime prediction model was selected. To choose an appropriate runtime equation, the benchmark was classified into a dwarf class. At the final step, the runtime of the workload was estimated.

The predicted runtimes of the applications were compared to the actual runtimes on the virtual machines of Amazon EC2. The virtual machines that we used in the experiment were the General purpose, Compute optimized, and Memory optimized instances (see Table 3.5 for more details). Although both experiments had the same testing steps, the presented results and the discussions were different.

In the following sections, we will henceforth call a sample of a profile of the benchmark as a 'sample data' or a 'sample profile'. The sample data was a set of MICA and Perf metrics that were collected in 1 minute-interval. The full-length run profile will be called a 'full-run data' or a 'full-run profile'. It is a profile that was collected from the beginning until the end of the execution.

4.1 Experiment on Trained Benchmarks

The purpose of this experiment was to verify that the sample data could be used instead of the full-run data. In the models construction phase, we trained the models by using the full-run data. However, we used the sample profile in the experiments. Therefore, we needed to ensure that the sample data would give the accurate runtime prediction results as the full-run data.

In the experiment, two types of benchmarks, type A and type B, were selected from each dwarf. The two benchmarks were the same application, but had different input sizes. Type A and type B represented a small input size and a large input size, respectively. Table 4.1 shows the actual runtimes of the selected benchmarks. In the discussion, the actual runtimes will be compared to the predicted runtimes.

Table 4.1 Actual Runtimes of Trained Benchmarks

Dwarf: Benchmark	Type	Actual Runtime (second)		
		General Purpose	Compute Optimized	Memory Optimized
dense: *nn*	*A*	3081	2492	1822
	B	15841	7572	5822
sparse: *cg*	*A*	233	159	140
	B	694	438	388

Dwarf: Benchmark	Type	Actual Runtime (second)		
		General Purpose	Compute Optimized	Memory Optimized
spectral: *spectral*	*A*	111	74	73
	B	286	186	177
nbody: *nbody2d*	*A*	894	785	592
	B	11291	9348	7020
sgrid: *particle*	*A*	2168	580	883
	B	96305	14174	26326
mapred: *monteCarlo*	*A*	3067	2022	1193
	B	12133	12457	2292
grapht: *quickSort*	*A*	370	245	219
	B	708	507	432

Before using the sample profiles for predicting the runtime of the benchmarks, we plotted the Kiviat diagrams to determine the similarity between the sample and the full-run data. The values plotted in the graphs are normalized as the Z-scores, equation (3.1). For each graph, the red polygon and the blue polygon represent the plot of a sample profile and a full-run profile, respectively. The diagrams are shown in Figure 4.1.

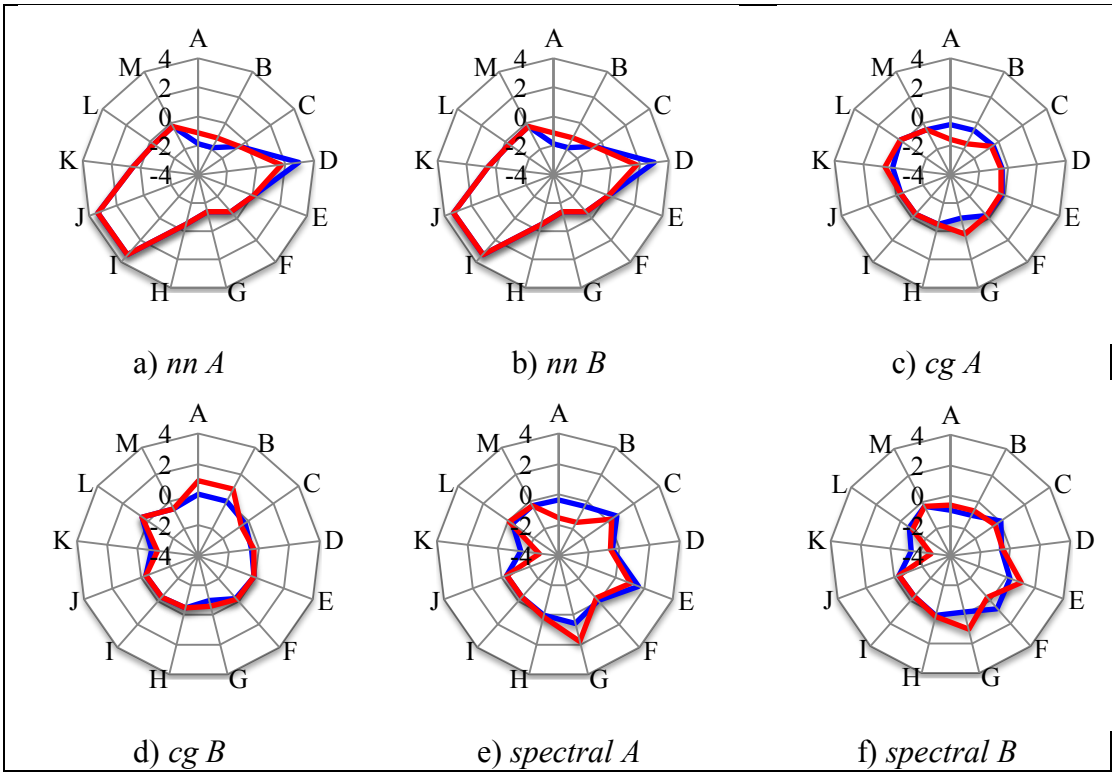

Figure 4.1 Kiviat Diagrams of Sample and Full-length Run Data

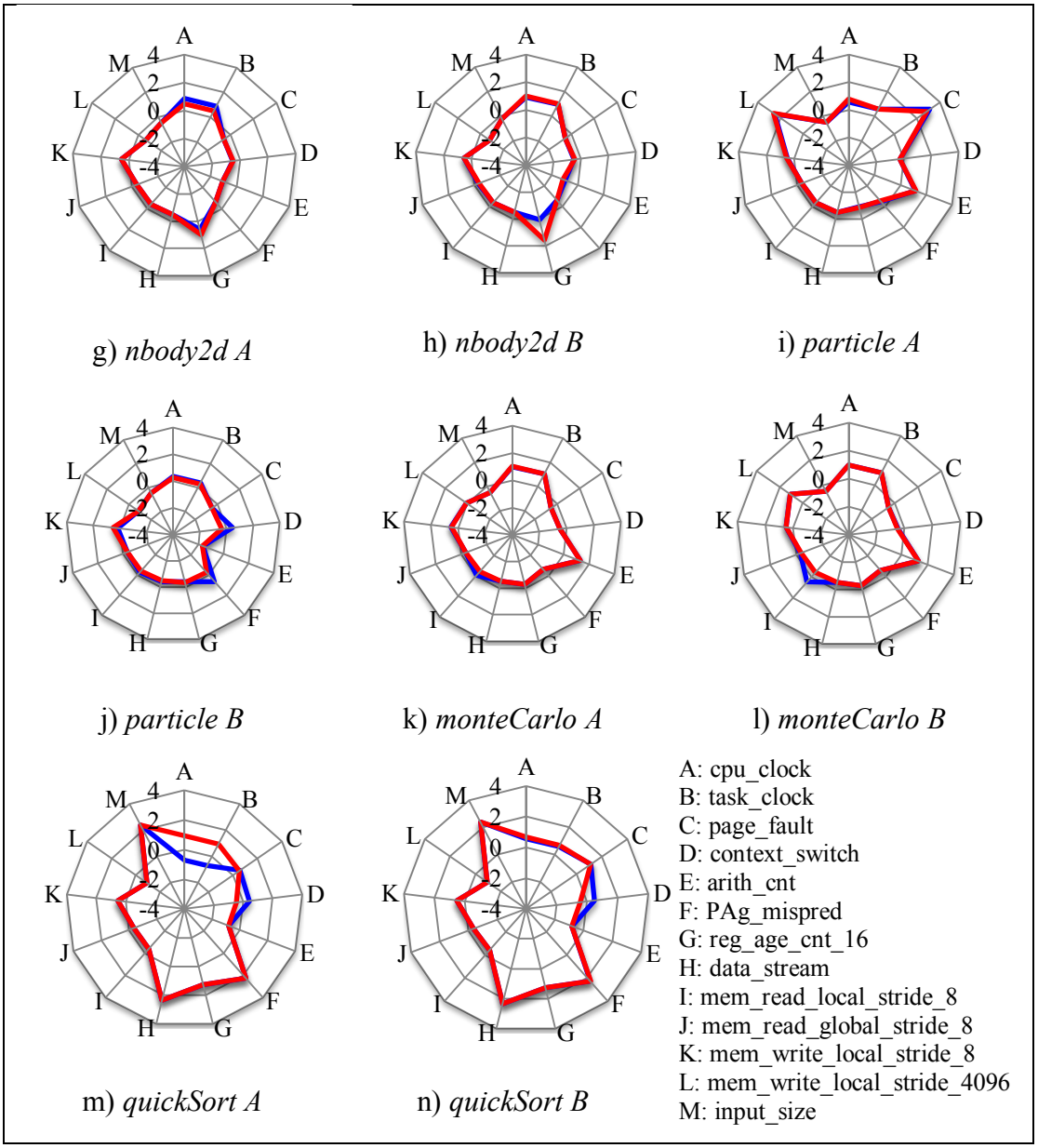

Figure 4.1 Kiviat Diagrams of Sample and Full-run Data

For each diagram in Figure 4.1, the red line (which represents a sample data) almost conceals the blue line (which represents a full-run data). This is apparent that the sample data and the full-run data were approximately the same. Therefore, the sample data can be used to represent the full-run data. Furthermore, we believe that this conclusion can be applied to other benchmarks as well.

The discussion of the experiment is divided into two parts: the workload classification and the runtime prediction.

4.1.1 Discussion on Workload Classification

The sample data was entered to the workload classification model. The correctness of the classification is shown in Table 4.2.

Table 4.2 Classification Results of Trained Benchmarks

Benchmark	Dwarf	Classified As
nn A	*dense*	*dense*
nn B		*dense*
cg A	*sparse*	***sgrid***
cg B		*sparse*
spectral A	*spectral*	*spectral*
spectral B		*spectral*
nbody2d A	*nbody*	*nbody*
nbody2d B		*nbody*
particle A	*sgrid*	*sgrid*
particle B		*sgrid*
monteCarlo A	*mapred*	*mapred*
monteCarlo B		*mapred*
quickSort A	*grapht*	*grapht*
quickSort B		*grapht*

All benchmarks, except *cg A*, were correctly classified into classes. We also investigated the error in *cg A* classification. As shown in Table 4.2, *cg A* was categorized into *sgrid*. We noticed that the sample data of *cg A* contained one metric that was irregular. The value of the *mem_write_local_stride_4096* metric was higher than the mean value, which could cause an error in classification. However, in the next step, we used both *sparse* and *sgrid* runtime prediction models for *cg A*.

4.1.2 Discussion on Runtime Prediction

The sample data was also used in runtime prediction. The model for each workload was selected based on the classification results in the previous step. However, a runtime of the *cg A* application was predicted by two models: *sparse* and *sgrid*. The prediction results are presented in Table 4.3. Moreover, the comparisons between the prediction results and the actual runtime for general purpose, compute optimized, and memory optimized virtual machines are visualized in Figure 4.2, Figure 4.4, and Figure 4.4, respectively.

Table 4.3 Runtime Prediction Results for Trained Benchmark

Benchmark	General Purpose		Compute Optimized		Memory Optimized	
	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)
nn A	3501.94	13.67	3168.96	27.17	2131.8	17
nn B	15559.81	1.76	7068.03	6.66	5801	0.36
cg A *sparse* model	253.31	8.71	208.30	31	145.23	3.73
cg A *sgrid* model	5270.46	> 100	27971.21	> 100	5317.04	> 100
cg B	758.16	9.4	495.68	13.17	384.03	0.77
spectral A	101.8	8.29	98.92	**33.67**	85.18	16.69
spectral B	278.73	2.54	196.87	5.84	203.66	15.06

Benchmark	General Purpose		Compute Optimized		Memory Optimized	
	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)
nbody2d A	1062.31	18.83	839.31	6.91	800.23	**35.17**
nbody2d B	12557.56	11.22	9528.01	1.92	8293.39	18.14
particle A	2477.22	14.26	701.06	20.87	801.73	9.20
particle B	76925.63	20.12	16133.74	13.82	18765.20	28.72
monteCarlo A	3333.45	8.69	2517.61	24.51	1173.10	1.67
monteCarlo B	9420.92	22.35	8729.66	29.92	2210.49	3.56
quickSort A	444.88	20.24	296.08	20.85	296.76	**35.51**
quickSort B	500.41	29.32	349.24	31.12	366.34	15.19

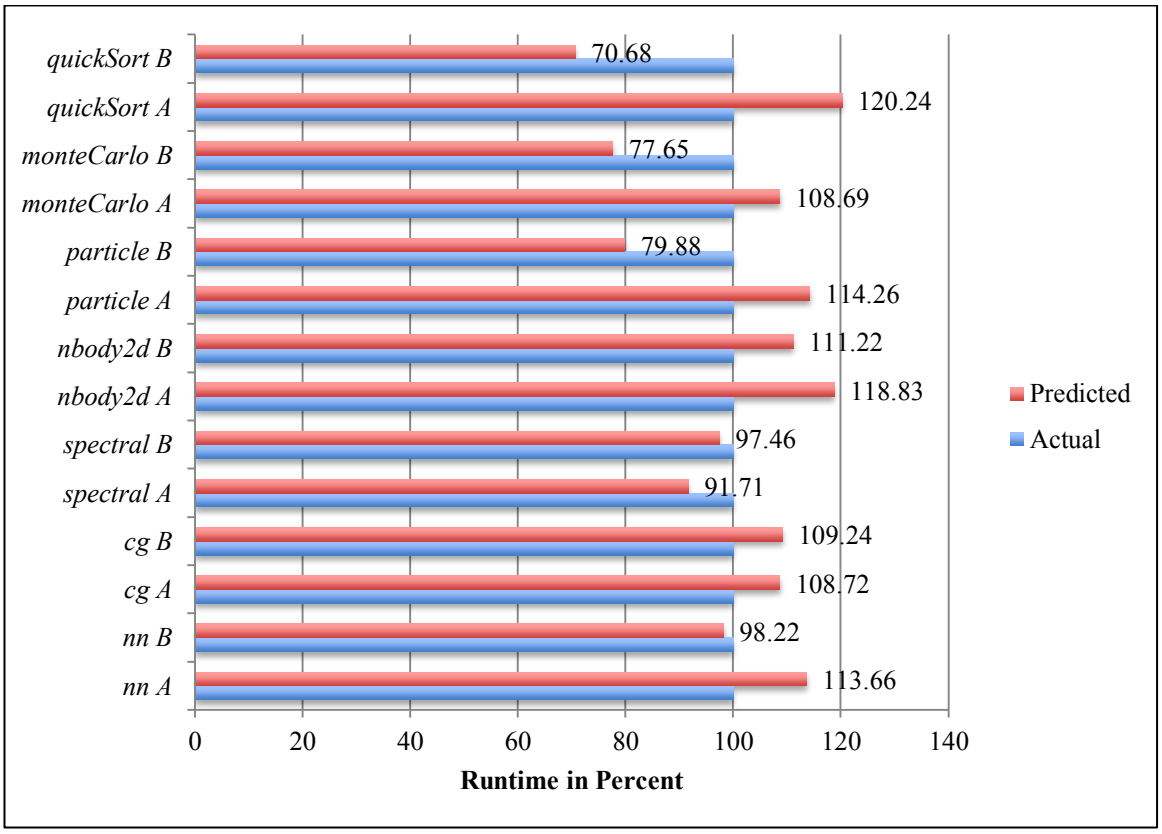

Figure 4.2 Runtimes of Trained Benchmarks in Percent for General Purpose Machine

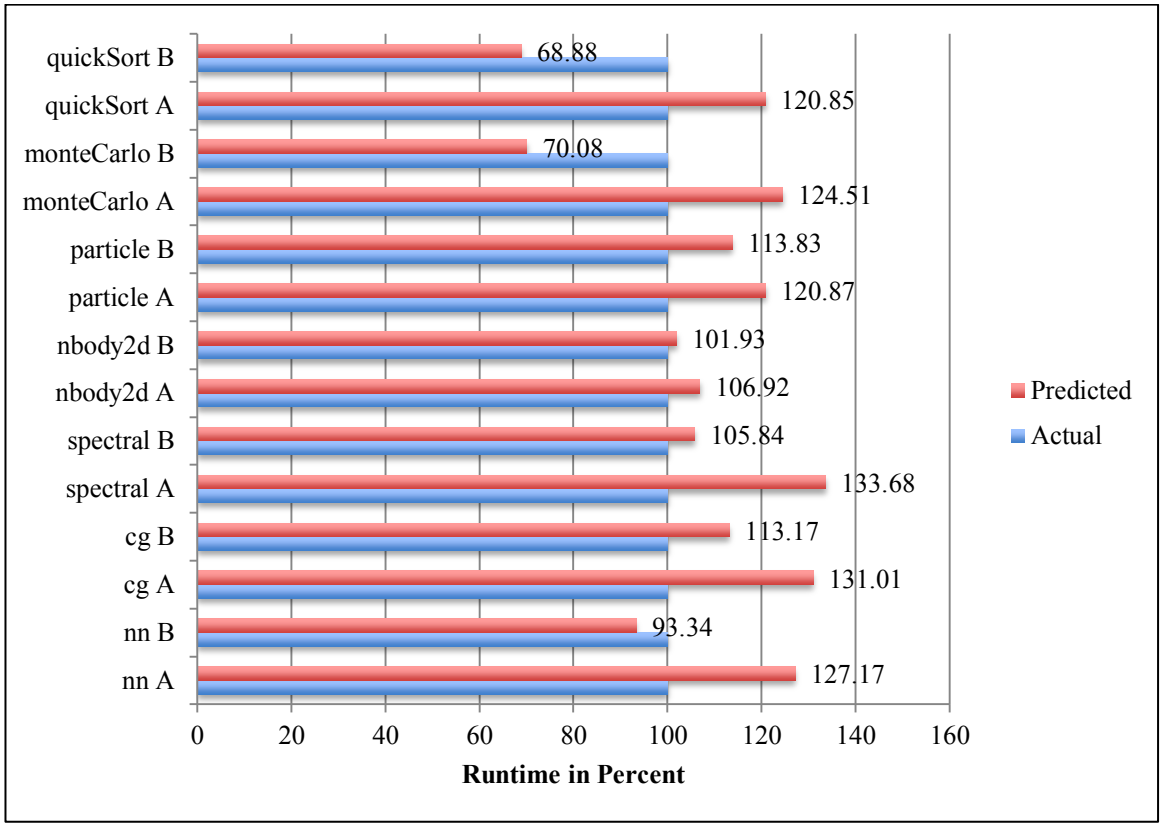

Figure 4.3 Runtimes of Trained Benchmarks in Percent for Compute Optimized Machine

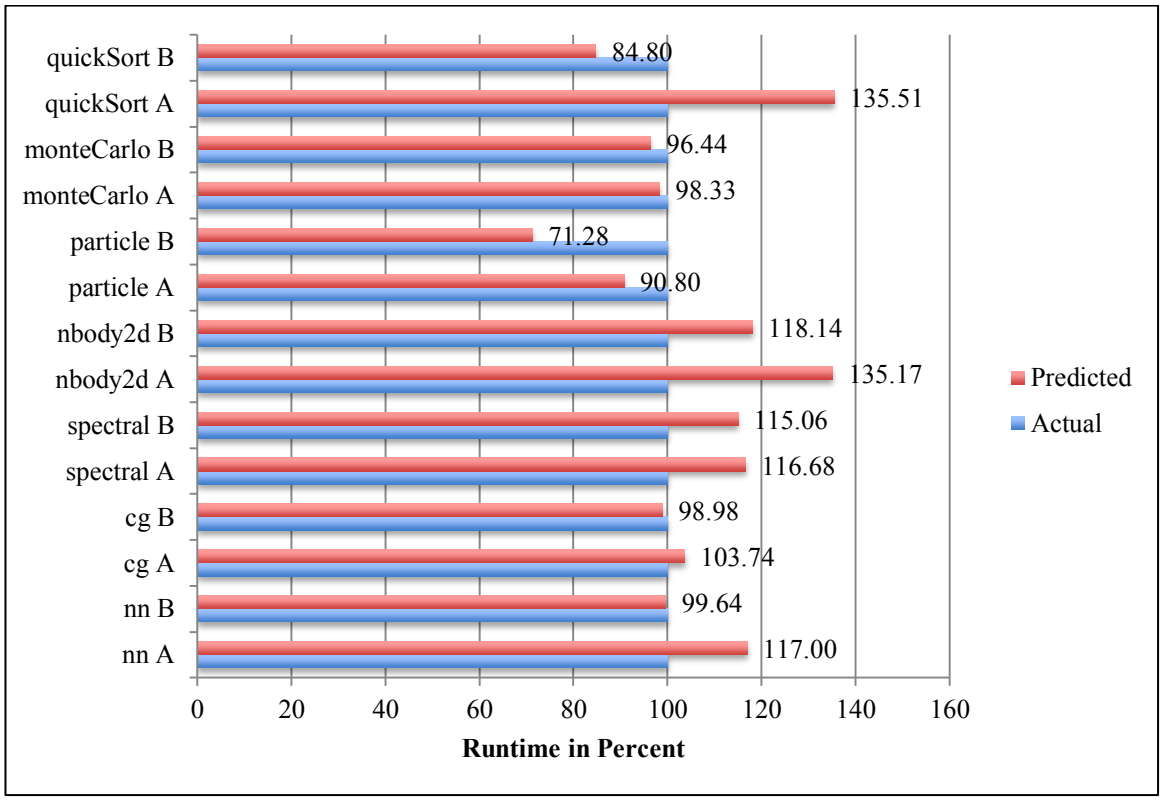

Figure 4.4 Runtimes of Trained Benchmarks in Percent for Memory Optimized Machine

From the experimental results, the maximum and minimum errors for the runtime prediction were 0.36% and 35.51%, respectively. The overall results were acceptable, however; we examined the causes of the errors that higher than 30%. There were three benchmarks that obtained high runtime prediction error: *spectral A*, *nbody2d A*, and *quickSort A*.

From the investigation, the runtime prediction models for the three benchmarks could yield the highly satisfactory estimation results for the benchmarks with a longer running period. In the model-training step, the number of data records that contained long-runtime was significantly more than those with short-runtime because our framework was proposed for the HPC applications, which generally have a long execution time. Therefore, the models might be more appropriate for predicting the runtime of the HPC applications with high execution time.

Moreover, for the runtimes of *cg A* that was predicted by the *sgrid* model were completely incorrect. This might imply that our framework was sensitive to the outliner. In order to improve the strength of classification and runtime prediction models, more data are needed in the training step.

4.2 Experiment on Untrained Benchmarks

This experiment aimed to verify the practicality of our framework. We would verify that our framework could be used to accurately predict the runtime of untrained applications. Five distinct untrained applications, from three benchmark suites, were adopted in the experiment. The benchmark suites were BioInfoMark [29], Galois [30], and LAPACK [31]. These benchmarks have been frequently used in the experiments of many researches [13][32][33].

BioInfoMark is a benchmark suite that contains bioinformatics applications. It has been developed by the IDEAL lab at the University of Florida. In our experiment, only three applications, which are *clustalw*, *glimmer*, and *predator*, were employed.

Galois is a framework for executing the serial C++ application on the shared memory machines. It is a work of the University of Texus at Austin. We adopted one of the applications in the Galois suite in our experiment, which is *nbody*.

LAPACK is a short name of Linear Algebra PACKage. It is a well-known library written in Fortran 90 for solving linear algebra system. It is provided by the University of Tennessee; the University of California, Berkeley; the University of Colorado Denver; and NAG Ltd. We implemented one dense linear algebra application from the LAPACK library. It performs lower-upper matrix factorization. We will call this application '*lu*'.

Notice that the selected benchmarks are representative of popular scientific applications.

In order to utilize the benchmarks in our experiment, we complied each benchmark by using gcc-4.4 on Ubuntu 12.04. The sample profiles of the benchmarks were collected on the master computer (see Table 3.5 for more details). The actual runtimes were measured by running the benchmarks on 3 types of virtual machines on the cloud: General purpose, Compute optimized, and Memory optimized. For each actual runtime,

we calculated it from the average runtime of three replicated runs. The summarized details of the benchmarks and the actual runtimes are shown in Table 4.4.

Table 4.4 Actual Runtimes of Untrained Benchmarks

Benchmark	Suite	Actual Runtime (second)		
		General Purpose	Compute Optimized	Memory Optimized
clustalw	BioInfoMark	200	152	121
glimmer		211	158	127
predator		269	202	161
nbody	Galois	3132	2225	1734
lu	LAPACK	23357	16647	10959

Similar to the previous experiment, the sample profiles (MICA and Perf metrics) of the benchmarks were used in workload classification and runtime prediction. The classification and runtime prediction results are shown in Table 4.5. The bar graphs that compare the predicted runtimes against the actual runtimes are shown in Figure 4.5, Figure 4.6, and Figure 4.7.

Table 4.5 Runtime Prediction Results for Untrained Benchmarks

Benchmark: Class	General Purpose		Compute Optimized		Memory Optimized	
	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)	Predicted Runtime (s)	Error (%)
clustalw: dense	231.49	15.74	201.54	**32.59**	177	**46.26**
glimmer: dense	286.61	**35.83**	224.89	**42.33**	114.07	10.18
predator: sparse	217.87	19.01	126	**37.49**	142.63	11.41
nbody: nbody	2764.49	11.73	1610.69	27.61	1509.01	12.98
lu: dense	23362.96	0.02	18183.42	9.23	11700.13	6.47

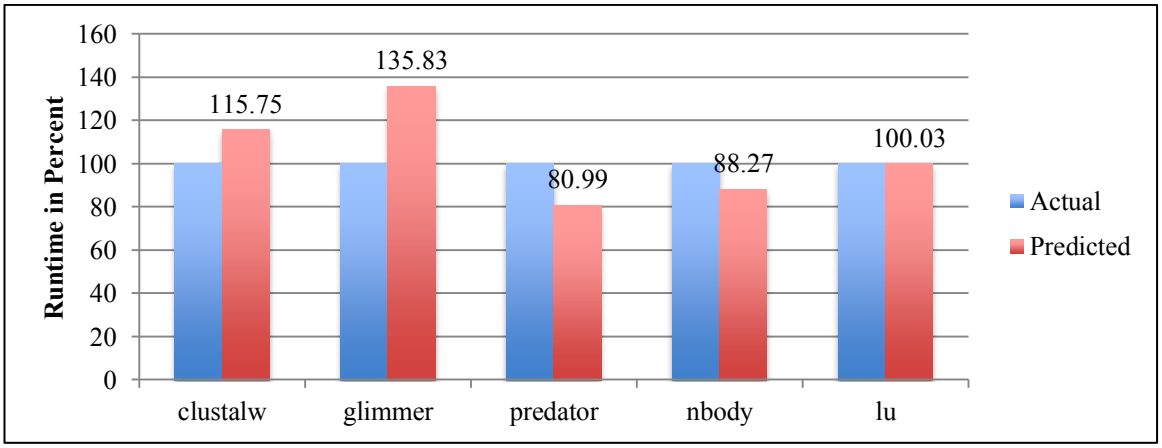

Figure 4.5 Runtimes of Untrained Benchmarks in Percent for General Purpose Machine

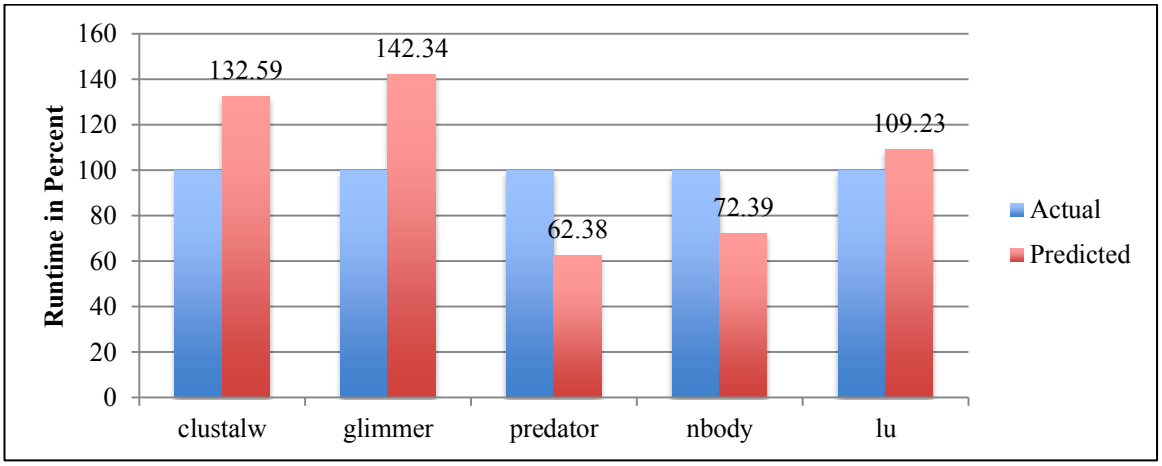

Figure 4.6 Runtimes of Untrained Benchmarks in Percent for Compute Optimized Machine

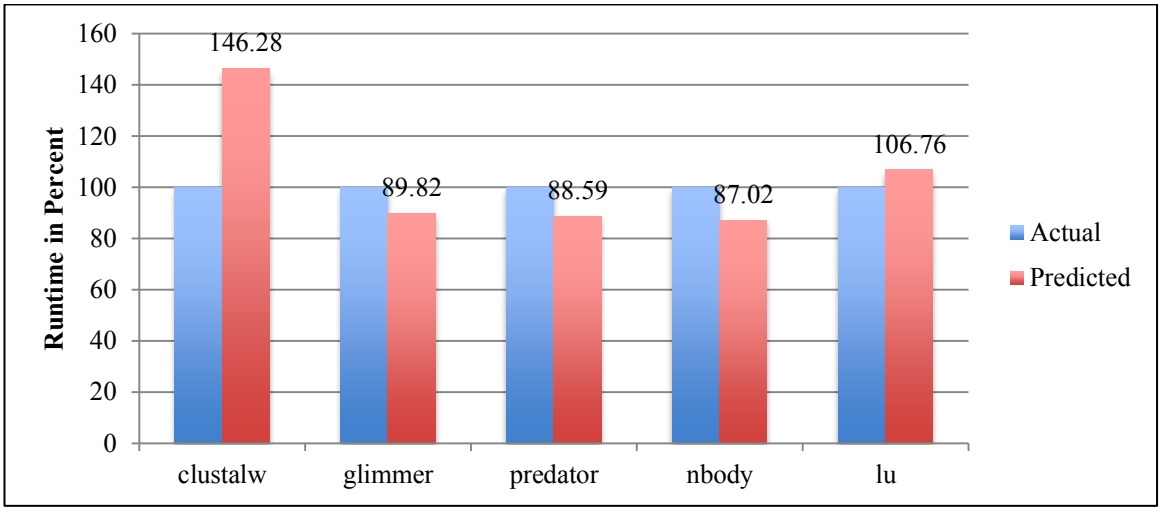

Figure 4.7 Runtimes of Untrained Benchmarks in Percent for Memory Optimized Machine

From Table 4.5, all the errors of runtime prediction for the benchmarks in the Galois and LAPACK suites were lower than 30%. Moreover, almost all predicted runtimes for *nbody* and *lu* were highly accurate with the highest error of 13%. Evidence to support the experimental results was the similar applications, which were *nbody2d*, *lu (A, B, C, S)*, and *lud*, that were used to train the models in the models construction phase.

For the remaining benchmarks in the BioInfoMark suite, the prediction results were not as good as expected. Most of the errors were higher than 30%. After inspecting the flaws in our models, we found that there were not sufficient similar benchmarks trained in our framework. One of the runtime prediction approaches [34] obtained the runtime estimation errors between 35% and 70%. They observed that the poor estimation results were caused by the insufficiency of the similar jobs in the historical data. In addition, as discussed in the previous experiment, the model might be more suitable for estimating the runtimes of workloads that have longer execution times.

Our average runtimes for General purpose, Compute optimized and Memory optimized virtual machines were 16.46%, 29.85%, and 17.46%, respectively. From a similar piece of research [7], they reported that the average runtime estimation errors of their

approach were approximately 45%, and the framework could significantly improve the overall performance of grid. Therefore, the error percentages in our runtime prediction would be acceptable. As a result, the predicted runtimes are accurate enough to allow the scheduler to be more efficient.

In conclusion, our framework can be used for predicting the runtime of the workloads that have similar execution pattern or characteristic with the trained workloads. Therefore, the decision tree and the models that are presented in this work can only be used for the specific group of applications, which includes the applications that have the similar data transfer and execution patterns as the 7 Berkley's Dwarfs (*dense*, *sparse*, *spectral*, *nbody*, *sgrid*, *mapred*, and *grapht*). However, our methodology is apparently better than the user-estimated approach because the runtimes given by users always yield more than 50% error [4].

To improve the practicality of the models, they are required to be trained extensively. More data records, for wide range of applications, are needed to be imported to the models.

In order to verify the practicality of our framework, the details on the adoption of our approach in the ALICE O2 project at CERN will be presented in the section. The traning applications as well as the testing benchmarks will be the real programs used in the ALICE experiment.

CHAPTER 5 THE ADOPTION OF RUNTIME PREDICTION FRAMEWORK IN ALICE O2 PROJECT

Through the collaboration between the Computer Department at King Mongkut's University of Technology Thonburi (KMUTT) and the ALICE's Computer Working Group at The European Council for Nuclear Research, or CERN, we were able to apply our framework to predict the runtime of real scientific applications from the ALICE experiment. This section provides details of the adoption of our runtime prediction framework, which is a part of the ALICE O2's scheduler. ALICE stands for A Large Ion Collider Experiment. It is one of 4 detectors of a Large Hadron Collider (LHC) at CERN. ALICE's main mission is to study about the highly interacting matters and the quark-gluon plasma. ALICE O2 is a project of the computer working group under the ALICE experiment. In fact, ALICE O2 is divided into several subgroups. Our study involved in only the data acquisition group (DAQ), responsible for receiving the collision events from the detectors and storing them in storages for further processing. In order to handle big data, several ten applications were used in ALICE's computer system.

In the year 2018, the ALICE detectors will be upgraded and the amount of data that will be produced from the detectors will be higher. The data throughput will be approximately 1 TB per second. For this reason, the whole ALICE's system has to be developed. The data flow of the ALICE experiment is shown in Figure 5.1.

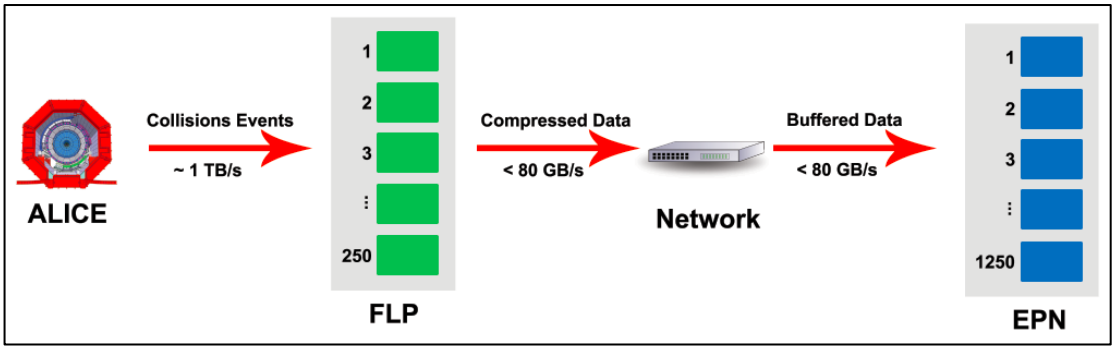

Figure 5.1 Data Flow of ALICE

The constraint of the new ALICE experiments is the system should be able to handle a large amount of particles collision events that will be produced from detectors. The data acquisition comprises two computer clusters – First Level Processors (FLP) and Even Processing Nodes (EPN). The two clusters serve different purposes. The FLP cluster is responsible for controlling the data rate to not exceed 80 GB/s peak and 20 GB/s average because the data have to be streamed to a permanent storage. Then, the data will be sent through the network to EPNs for further processing, for example, event reconstruction.

The mission of the ALICE O2 project is to select the suitable computing platforms, both FLPs and EPNs, by optimizing the size and cost of an online farm. Various platforms have been tested out using the benchmark algorithm called 'Pixel Cluster Finder'. Moreover, the scheduler between the two clusters is required in order to efficiently propagate jobs and data from FLPs to EPNs. The scheduler is placed in the network between FLPs and EPNs as illustrated in Figure 5.2.

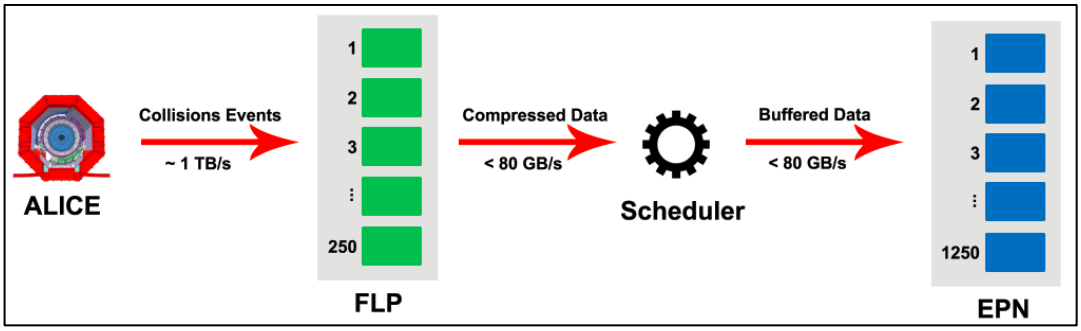

Figure 5.2 The Position of Scheduler

In order to schedule jobs from FLPs to EPNs, two challenges arise:

1. The scheduler should be able to accurately calculate the runtime of jobs on different computers because this information will be used in scheduling process.
2. The scheduler should work fast and efficiently because the delay due to the jobs scheduling may cause *severe processing bottleneck*, and provoke side effects like increasing the buffer space needed on FLPs.

Our framework was adopted to satisfy the first challenge mentioned above. The runtime estimation process was integrated to the scheduler to predict runtimes of the applications in the system. The estimated runtime would be used further in the scheduling process.

In this work, complexity analysis was also used in order to find an appropriate runtime estimation equation. It can be used directly only in the case that a source code of the application is provided. An application with source code will be henceforth called a 'white-box' application. Notice that the complexity analysis has to be done by hand. In addition, a runtime equation for each application has to be constructed separately using linear regression. In this approach, a runtime prediction equation represents a relationship between inputs and a runtime, for example, $runtime = f(x_i)$ where x_i is an input parameter. Moreover, it requires input parameters to calculate the application's runtime. A method to build a runtime prediction equation using this approach will be presented in the next subsection.

In a black-box application, the runtime estimation equations are determined by using the method that was presented in Chapter 3. In order to estimate the runtime of an application, the profile of the application will be sampling for a certain period by using MICA and Perf tools. After that, it will be categorized into a dwarf class, and the classification results will be used to select the runtime prediction model. In the final step, the runtime in second unit will be computed by using the sample profile as the input of the runtime estimation model.

The following subsection presents the initial experiment and the results from the adoption of our runtime estimation framework in the ALICE O2 project. The experiments were divided into two parts: the experiment on white-box applications adopting a complexity analysis and linear regression to create runtime equations and the experiment on black-box applications employing a workload classification, ABC and linear regression to construct runtime prediction models.

5.1 Initial Experiment and Result for White-Box Applications

In this section, we applied the complexity analysis technique in collaboration with linear regression to determine the runtime prediction model for specific applications with a source code. This method was a contribution from our previous work [35]. The method for constructing the runtime prediction equation is shown in Figure 5.3.

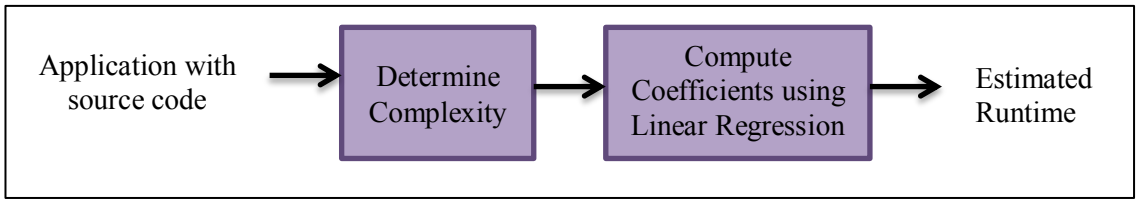

Figure 5.3 Runtime Estimation Model Construction for Known-Source Code Application

In order to build the runtime estimation model for a white-box application, a complexity of the algorithm has to be determined. Application's complexity is a mathematical equation that represents a relationship between input attributes and runtime of the application: *runtime* = $F(x_1, x_2, x_3, ..., x_n)$ where x_i is an input attribute. After the model is obtained, an empirical experiment is run in order to get the data records to fit the model. Then, the model is fitted with the collected data by using linear regression, and coefficients will be obtained as an output.

In this experiment, we adopted TPC laser events program, with a known source code. According to the method described above, the complexity of the algorithm is determined, and the result is shown in equation (5.1).

$$R = \beta_1 S + \beta_0 \tag{5.1}$$

where R is an estimated runtime
S is an input size in megabyte (Mb)
β is coefficient of each variable and i is number of variable

Then, we ran TPC laser events by adjusting input sizes in order to get 500 different runtime values. Notice that the runtime of each input size was an average of 5 replicated runs. The experiment was run on the Scientific Linux CERN 6 (SLC6) operating system. The computer used in the experiment had 1 Intel Core i5-4570 CPU @ 3.20GHz, 8 GB memory, and 1 TB storage.

After fitting the model using linear regression, we obtained a complete model as shown in equation 5.2.

$$R = 0.0713S + 4.93 \tag{5.2}$$

The quality of the regression model was measured by R-squared. This model obtains 99% R-squared, which is as high as expected. We plotted a graph between input size and actual runtime, as shown in Figure 5.4, and we found that they are linearly correlated. Therefore, it is not surprising that our runtime prediction model, equation 5.1, could achieve a high R-squared value.

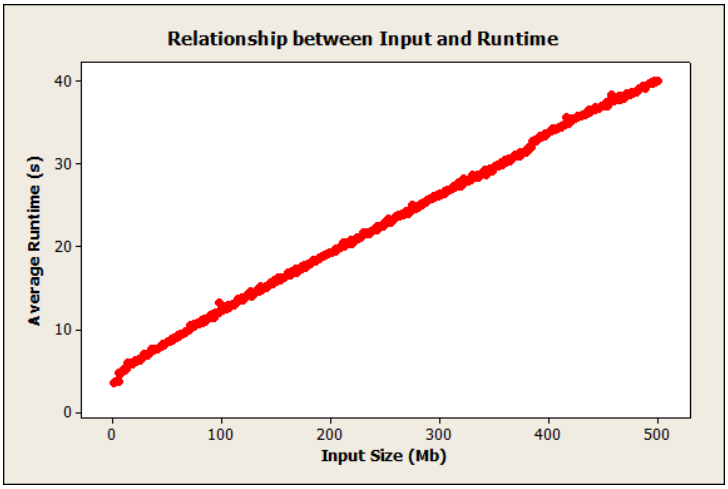

Figure 5.4 Linear Relationship between Input and Runtime

In order to test the accuracy of the model, we ran the application with the different input sizes as shown in Table 5.1. Notice that the input sizes from 100 Mb to 500 Mb were used in model training, while the rest were not.

Table 5.1 Results from Running TPC Laser Events with Different Input Sizes

Input Size (Mb)	Predicted Runtime (s)	Actual Runtime (s)	Error (%)
100	12.06	11.93	1.09
200	19.19	18.79	2.13
300	26.32	25.79	2.06
400	33.45	36.22	7.65
500	40.58	39.95	1.58
600	47.71	46.95	1.62
700	54.84	54.06	1.44
800	61.97	62.23	0.42

Given the input size, our mathematical model can accurately predict the runtime of application with less than 10% error. Even for untrained data, the runtimes can be predicted with high accuracy. To sum up, this model can be used for calculating the runtime of the TPC laser events application for both trained and untrained data.

5.2 Initial Experiment and Result for Black-Box Applications

This experiment used the black-box applications, source code not being provided; therefore, the method that was presented in chapter 3 was applied. In order to predict a runtime of an application, the workload classification model and the runtime estimation equations are required. However, we could not use the same models (as shown in chapter 3) because the operating systems and hardware of the machines were different. In this experiment, the machine contained 8-core Intel Core i7-2600 CPU, 8-GB memory, and 470 GB of storage, and it ran the SLC6 operating system. Thus, we needed to construct new workload classification model and runtime estimation equations.

To train the models, we collected the profiles of the benchmarks (shown in Table 3.4) by using MICA and Perf tools on a master machine. Notice that the master machine had

the same specification as the machine that used in the experiment. A profile of each benchmark was collected from the beginning to the end of the execution (full-run profile). For each class of dwarf, we collected 15 profiles where each profile contained 12 metrics. 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. As mentioned in section 3.1, the input attributes for the algorithm were only 8 MICA metrics. The rules derived from the decision trees are as follows:

Rules for *dense*

(B ≤ 0.00395 ^ E ≤ 0.994573 ^ H ≤ 0.000786 ^ A ≤ 0.663777) ∨
(B ≤ 0.00395 ^ E > 0.994573 ^ A ≤ 0.712571 ^ A > 0.606547 ^ H > 0.00002)

Rule for *sparse*

(B ≤ 0.00395 ^ E ≤ 0.994573 ^ H ≤ 0.000786 ^ A > 0.663777) ∨
(B > 0.00395 ^ C > 0.536309 ^ A > 0.635292 ^ F ≤ 0.000025 ^ A > 0.769438 ^ A ≤ 0.781248) ∨
(B > 0.00395 ^ C > 0.536309 ^ A > 0.635292 ^ F > 0.000025)

Rule for *spectral*

(B ≤ 0.00395 ^ E ≤ 0.994573 ^ H > 0.000786) ∨
(B > 0.00395 ^ C ≤ 0.536309 ^ B ≤ 0.011211)

Rule for *nbody*

(B ≤ 0.00395 ^ E > 0.994573 ^ A > 0.712571 ^ B ≤ 0.00158 ^ C ≤ 0.603917) ∨
(B > 0.00395 ^ C > 0.536309 ^ A ≤ 0.635292)

Rule for *sgrid*

(B ≤ 0.00395 ^ E > 0.994573 ^ A ≤ 0.606547) ∨
(B ≤ 0.00395 ^ E > 0.994573 ^ A ≤ 0.712571 ^ A > 0.606547 ^ H ≤ 0.00002) ∨
(B ≤ 0.00395 ^ E > 0.994573 ^ A > 0.712571 ^ B > 0.00158)

Rule for *mapred*

(E > 0.994573 ^ A > 0.712571 ^ B ≤ 0.00158 ^ C > 0.603917) ∨
(B > 0.00395 ^ C ≤ 0.536309 ^ B > 0.011211)

Rule for *grapht*

(B > 0.00395 ^ C > 0.536309 ^ A > 0.635292 ^ F ≤ 0.000025 ^ A ≤ 0.769438) ∨
(B > 0.00395 ^ C > 0.536309 ^ F ≤ 0.000025 ^ A > 0.781248)

Note:

A denotes *reg_age_cnt_16*
B denotes *PAg_mispred*
C denotes *arith_cnt*
D denotes *data_stream*
E denotes *mem_write_local_stride_4096*
F denotes *mem_write_local_stride_8*
G denotes *mem_write_global_stride_8*
H denotes *mem_read_loca_stride_8*

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 experiment, TPC laser event program was used. To build a runtime prediction equation, we collected the profiles of the TPC laser event with various input sizes and used them to train the model. Notice that we did not build a model for every dwarf class. We constructed only a model for the class that the application belonged to. From the rules presented above, TPC laser event was categorized into *dense*. Therefore, only *dense* runtime prediction equation was constructed. We applied ABC and linear regression to training data and derived the runtime equation, which can yield 99% R-squared, as shown in equation (5.3). Notice that notations used in the equation were referenced from Table 3.6.

$$runtime_{dense} = 14 + 0.254\ \mathrm{sqrt}(M) + 5075\ G + 60311\ I + 0.0441\ C - 79824\ K - 13.7\ \mathrm{sqrt}(A) + 153\ \log(K) - 11.1\ \mathrm{sqrt}(C) - 7104\ \mathrm{sqrt}(I) - 1949\ \mathrm{power}(E, 2) - 146\ \ln(H) \quad (5.3)$$

We predicted runtimes of the TPC laser event application with different input sizes by using equation 5.3. The predictive runtimes are shown in Table 5.2 and Figure 5.5.

Table 5.2 Runtime Prediction Results for Black-box Application

Input Size (MB)	Actual Runtime (s)	Predicted Runtime (s)	Error (%)	Note
977	80	64.15	19.82	Trained
1500	116	142.02	22.43	
1954	152	150.85	9.56	Trained
2000	154	168.72	0.76	
2500	193	243.45	26.14	
3000	229	281.17	22.78	
3500	266	261.51	1.69	
3909	296	315.03	6.43	Trained
4000	354	451.84	27.64	
5000	431	504.88	17.14	
7819	590	574.36	2.65	Trained
8796	691	626.01	9.41	

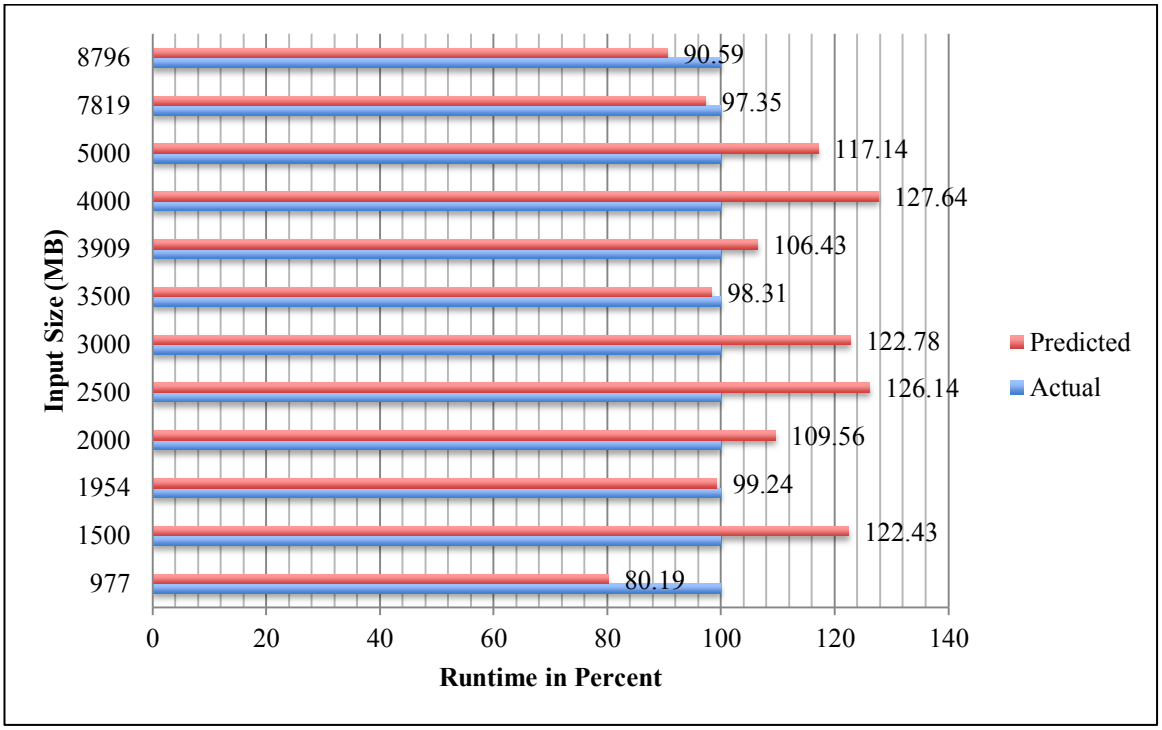

Figure 5.5 Runtimes of TPC Laser Events in Percent

From Table 5.2, the records that were marked 'Trained' were used in runtime equation training phase, while the rest were used only during the testing step. The overall runtime prediction results were accurate for both trained and untrained data. The prediction errors were approximately between 1% and 30%, which were acceptable.

Apart from TPC laser event, we also validated our framework against another application, namely 'PHS'. We applied the methods for both white-box and black-box applications on the data. The experiment procedure followed the same method as the one for previous application. For the black-box approach, the application was categorized into the *sparse* class. From the experimental results, the average runtime prediction errors for white-box and black-box methods were 3.87% and 4.31%, respectively. Although the average errors were not significantly different from each other, the runtime prediction errors for black-box applications were slightly higher than another approach. Moreover, the highest prediction errors for both methods were smaller than 10%, which was highly satisfying.

Both runtime estimation approaches presented in this chapter could be used effectively to predict runtimes of the applications of ALICE O2 project. In the case that source code is accessible, complexity analysis and linear regression would be applied to build a runtime estimation equation. This approach is more accurate than another method, but it requires hand calculation and a runtime equation is specific to the application and machine.

Moreover, runtime of an application with unknown source code can be also predicted. In the experiment, we assumed that source code of TPC laser event was not provided. We used dwarf benchmarks (see Table 3.4 for more details) to train both workload classification model and runtime prediction equation. In addition to these benchmarks, a few TPC laser event's profiles with different input sizes were also used for runtime

estimation model training. The accuracy of this model was much less accurate than the method for white-box applications, however; it was still acceptable. Furthermore, it was more practical because a runtime prediction model was only dependent to a specific class rather than the application. Therefore, the number of required runtime prediction equation could be reduced substantially.

To sum up, the method for white-box applications can predict a runtime with higher accuracy than the method for black-box applications. However, the source code must be provided, and a lot of manual processes are required. This method can fail easily when a new application is introduced to the system, which may seriously interrupt the system operation. In contrast, the runtime estimation method for black-box applications can predict the runtime of unknown applications automatically without having to train a new model. This property is important to a computer system that requires rapidity in job scheduling, like ALICE's system. Therefore, the runtime estimation framework for black-box applications is more efficient and more flexible when comes the real system.

CHAPTER 6 CONCLUSION

Cloud computing is a distributed environment platform that has received much attention during a past few years. Due to the characteristics of the cloud, the users are allowed to request the computer resources on demands, and the cloud will automatically allocate and de-allocate the resources upon on the requirement of the users. The resources in the cloud can be simply accessed via the Internet using the secure communication protocol, such as, secure shell (SSH) and remote desktop protocol (RDP). For the public cloud, the cost is usually calculated based on the extent of resources usage.

Since the cloud can provide the users with flexible and excessive computer resources, it is suitable for executing the high performance computing tasks. However, it is difficult to efficiently run the HPC applications in the cloud because there are many issues that must be considered. One of the most important issues is the method to accurately estimate the runtimes of the applications in the cloud because the runtime is required by most scheduling algorithm, for example, Backfilling and Heterogeneous Earliest Finish Time (HEFT). Inaccurate runtime estimation can degrade the performance of the scheduler, which could lead to the deterioration of the execution performance.

For this reason, we have presented the mechanism to estimate the runtimes of the applications with unknown-profile on the cloud. 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 other works. We utilized 12 performance metrics, measured by MICA and Perf tools, rather than using a user name and a project name. MICA and Perf metrics can represent the real characteristics of the applications. These attributes were used in both workload classification and runtime prediction phases.

The workload classification is the step to categorize an application of interest into a class that has similar execution behavior. The results from this step were used to choose the appropriate runtime prediction equation in the next step. The classes were specified based on the characteristic of the programs that were explained in Berkley's Dwarfs definition. There are 7 classes, *dense, sparse, spectral, nbody, sgrid, mapred,* and *grapht,* of which labels come from the names of dwarfs. In our work, we used a decision tree to classify the workloads. The input parameters of the decision tree were 8 MICA metrics, and the output was the name of class that the workload belonged to. The decision tree could yield 96.89% percent accuracy on a stratified 10-fold cross-validation.

After the application was already classified, the classification results were used to select the proper runtime estimation equation because the applications in the different classes had separated equation for estimating the runtime. In addition, the runtime estimation equation was also specific to the type of virtual machines that application would be run on. As shown in chapter 3, there were 21 runtime prediction equations presented because we had 3 types of virtual machines (General purpose, Compute Optimized, and Memory Optimized instances) and each machine needed 7 equations. In our framework, we built the runtime equations by using ABC and linear regression to find the structure and the coefficients of each equation. The input variables of a runtime equation were 12 performance metrics (MICA and Perf metrics) and the input size, which could be computed by the method presented in section 3.3.1. The output of the equation was a

predicted runtime. In order to measure the quality of the equations, R-squared was used. From the test, all equations could achieve high R-squared with at least 82.6%.

In the experiments, we tested our framework against 2 objectives. First, we used the trained benchmarks to verify that the sample profile (which is the set of MICA and Perf metrics that are collected for 1 minute) could be used instead of the full-run profile (the metrics collected from the beginning to the end of the execution). Second, we adopted the untrained benchmarks in order to ensure that our framework could be also applied to the untrained data. The actual runtimes for both experiments were collected on 3 instances of Amazon EC2: General purpose, Compute optimized, Memory optimized.

For the experiment on the trained benchmarks, the runtime prediction accuracy for most benchmarks was higher than 64.49%. We investigated that our framework yielded quite high error percentages in some cases that the actual runtimes were small. This could be caused from the insufficient number of small runtime applications in the training data. However, we can conclude that the sample data can replace the full-run data effectively.

In the untrained benchmarks experiment, we adopted the representative of the scientific applications from three benchmark suites that has been used widely in many researches: BioInfoMark, Galois, and LAPACK. The accuracy percentages of the benchmarks that had the similar characteristics to some benchmarks in the training data were high (less than 13% error). In contrast, the benchmarks that were not similar to the training benchmarks obtained high error percentages, which were between 31 and 47. This problem could be solved by adding similar jobs to the training data. However, the errors are acceptable since the presence of the higher error in the similar works could improve the execution performance significantly [7].

With our runtime prediction framework, runtimes of ALICE's applications can be accurately predicted. This would subsequently benefit to a task scheduler who requires application runtimes to determine a schedule of jobs. Since the system performance relies on the job scheduler, we believe that our method can improve the overall performance of ALICE's computer system.

To sum up, our runtime estimation framework can be used efficiently to predict the runtime of the unknown application in the cloud computing environment. With our approach, the scheduler will be able to generate high quality job schedules. Moreover, this method is more efficient than the user estimation approach. For the future work, we will run more extensive empirical experiments to collect more training data so that we can improve the accuracy of the runtime prediction model. In addition, the scheduling framework on the cloud should be developed so that we can obtain an integrated system that can efficiently predict the runtime and schedule the workloads on the cloud.

REFERENCES

[1] Baraglia, R., Capannini, G., Pasquali, M., Puppin, D., Ricci, L., and Techiouba, A. D., 2008, "Backfilling Strategies For Scheduling Streams Of Jobs On Computational Farms", **Making Grids Work**, pp. 103-115.

[2] Wieczorek, M., Prodan, R., and Fahringer, T., 2005, "Scheduling Of Scientific Workflows In The ASKALON Grid Environment", **ACM SIGMOD Record Journal**, Vol. 34, No. 3, pp. 56-62.

[3] Srinivasan, S., Kettimuthu, R., Subramani, V., and Sadayappan., 2002, "Characterization Of Backfilling Strategies For Parallel Job Scheduling", **Proceeding of the International Conference on Parallel Processing Workshop**, pp. 514–519.

[4] Tang, W., Desai, N., Buettner, D., and Lan, Z., 2013, "Job Scheduling With Adjusted Runtime Estimates On Production Supercomputers", **Parallel and Distributed Computing Journal**, Vo. 73, No. 7, pp. 926-938.

[5] Krishnaswamy, S., Loke, S. W., and Zaslavsky, A., 2004, "Estimating Computation Times Of Data-Intensive Applications", **IEEE Distributed Systems Online**, Vol. 5, No. 4.

[6] Smith, W., Foster, I., and Taylor, V., 2004, "Predicting Application Run Times With Historical Information", **Parallel Distributed Computing Journal**, Vol. 64, No. 9, pp. 1007 - 1016.

[7] Xia, E., Jurisica, I., Waterhouse, J., and Sloan, V., 2010, "Runtime Estimation Using The Case-Based Reasoning Approach For Scheduling In A Grid Environment", **Case-Based Reasoning. Research and Development**, pp. 525-539.

[8] Zhanga, Y., Suna, W., and Inoguchib, Y., 2008, "Predict Task Running Time In Grid Environments Based On CPU Load Predictions", **Future Generation Computer Systems Journal**, Vol. 24, pp. 489-497.

[9] Porter, C., 2011, "IaaS Provider", **Harnessing Public Cloud In HPC: Are All Infrastructure Providers Created Equal?**, pp. 4-5.

[10] Amazon Web Services, Inc, 2014, "Instance Types", **Amazon Elastic Compute Cloud User Guide API Version 2014-06-15**, pp. 101-122.

[11] Evangelinos, C., and Hill, C., 2008, "Cloud Computing For Parallel Scientific HPC Applications: Feasibility Of Running Coupled Atmosphere-Ocean Climate Models On Amazon's EC2", **The first Workshop on Cloud Computing and its Applications (CCA'08)**.

[12] Netjinda, N., Sirinaovakul, B., and Achalakul T., 2012, "Cost Optimization In Cloud Provisioning Using Particle Swarm Optimization", **Proc. of ECTI-CON 2012**, pp. 1-4.

[13] Hoste, K., and Eeckhout, L., 2007, "Microarchitecture-Independent Workload Characterization", **IEEE Micro**, Vol. 27, No. 3, pp. 63-72.

[14] Asanovic K., Bodik R., Catanzaro B.C., Gebis J.J., Husbands P., Keutzer K., Patterson D.A., Plishker W.L., Shalft J., Williams S.W., and Yelick K.A., 2006, "The Landscape Of Parallel Computing Research: A View From Berkeley", **Technical Report**, UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley.

[15] Feng, W., Lin, H., Scogland, T. and Zhang, J., 2012, "OpenCL And The 13 Dwarfs: A Work In Progress", **Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering**, ACM, pp. 291-294.

[16] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K., 1991, "The NAS Parallel Benchmarks - Summary And Preliminary Results", **International Journal of High Performance Computing Applications**, Vol. 5, No. 3, pp. 63-73.

[17] Bailey, D., Barszcz, E., Dagum, L., Frederickson, P., Schreiber, R., and Simon, H., 1994, "The Kernel Benchmarks", **RNR Technical Report**, RNR-94-007.

[18] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Sang-Ha Lee and Skadron, K., 2009, "Rodinia: A Benchmark Suite For Heterogeneous Computing", **IEEE International Symposium on Workload Charaterization**.pp. 44-54.

[19] Manakul, K., Siripongwutikorn, P., See, S. and Achalakul, T., 2012, "Modeling Dwarfs for Workload Characterization", **IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)**, pp. 776 – 781.

[20] Kaiser, A., Williams, S., Madduri, K., Ibrahim, K., Bailey, D., Demmel, J. and Strohmaier, E., 2010, "TORCH Computational Reference Kernels: A Testbed For Computer Science Research", **Technical Report**, UCB/EECS-2010-144. Electrical Engineering and Computer Sciences, University of California at Berkeley.

[21] Yu, J., Buyya, R. and Ramamohanarao, K., 2008, "Workflow Scheduling Algorithms For Grid Computing", **Proc. of Metaheuristics for Scheduling in Distributed Computing Environments**, Vol. 146, pp. 173-214.

[22] Udomkasemsub, O., Xiaorong, L., and Achalakul, T., 2012, "A Multiple-Objective Workflow Scheduling Framework For Cloud Data Analytics", **Proc. of the 9th International Joint Conference on Computer Science and Software Engineering (JCSSE'12)**, pp. 392-399.

[23] Vivekanandan, K., Ramyachitra, D. and Anbu, B., 2011, "Artificial Bee Colony Algorithm For Grid Scheduling", **Journal of Convergence Information Technology**, Vol. 6, No. 7, pp. 328 – 339.

[24] Banharnsakun, A., Achalakul, T., and Sirinaovakul, B., 2011, "The Best-So-Far Selection In Artificial Bee Colony Algorithm", **Applied Soft Computing Journal**, Vol. 11, No. 2, pp. 2888-2901.

[25] Karaboga, D. and Basturk, B., 2008, "On The Performance Of Artificial Bee Colony (ABC) Algorithm", **Applied Soft Computing**, Vol. 8, No. 1, pp. 687-697.

[26] Karaboga, D. and Akay, B., 2009, "A Comparative Study Of Artificial Bee Colony Algorithm", **Applied Mathematics and Computation**, Vol. 214, No. 1, pp. 108-132.

[27] Taetragool, U., and Achalakul, T., 2011, "Method For Failure Pattern Analysis In Disk Drive Manufacturing", **International Journal of Computer Integrated Manufacturing**, Vol. 24, No. 9, pp. 834-846.

[28] Chou, K. Y., Shih, C. C., Keh, H. C., Yu, P. Y., Cheng, Y. C., and Huang, N. C., 2013, "Using Decision Tree To Analyze Patient Of Aortic Aneurysm With Chronic Diseases In Clinical Application", **16th International IEEE Conference on Network-Based Information Systems (NBiS)**, pp. 405-409.

[29] Li, Y., and Li, T., 2005, "BioInfoMark: A Bioinformatic Benchmark Suite For Computer Architecture Research", **Technical Report**, IDEAL Research, ECE Dept., University of Florida.

[30] Pingali, K., Nguyen, D., Kulkarni, M., Burtscher, M., Hassaan, M. A., Kaleem, R., Lee, T., Lenharth, A., Manevich, R., Mendez-Lojo, M., Prountzos, D. and Sui, X., 2011, "The Tao Of Parallelism In Algorithms" **ACM SIGPLAN Notices**, Vol. 46, No. 6, pp. 12-25.

[31] Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammerling, S., Demmel, J., Bischof, C. and Sorensen, D., 1990, "LAPACK: A Portable Linear Algebra Library For High-Performance Computers", **Proceedings of the 1990 ACM/IEEE conference on Supercomputing**, pp. 2-11.

[32] Fu, X., Li, T., and Fortes, J., 2006, "Sim-Soda: A Unified Framework For Architectural Level Software Reliability Analysis", **Workshop on modeling, benchmarking and simulation**.

[33] Volkov, V., and Demmel, J. W., 2008, "Benchmarking GPUs To Tune Dense Linear Algebra", **Proceedings of the 2008 ACM/IEEE conference on Supercomputing**, p. 31.

[34] Li, H., Groep, D., and Wolters, L., 2005, "Efficient Response Time Predictions By Exploiting Application And Resource State Similarities", **Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing**, pp. 234-241.

[35] Pumma, S., Achalakul, T. and Xiaorong, L., 2012, "Automatic VM Allocation for Scientific Application", **Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems**, pp. 828-833.

CURRICULUM VITAE

NAME Sarunya Pumma

DATE OF BIRTH 2 August 1990

EDUCATION

2002 – 2007 High School Graduation, Major Science-Mathematics (GPA 3.72)
Satrinonthaburi School, 2002

2008 – 2011 Bachelor of Engineering (Computer Engineering) with the First Class Honor (GPA 3.84)
King Mongkut's University of Technology Thonburi, 2008

2012 – 2013 Master of Engineering (Computer Engineering) (GPA 4.0)
King Mongkut's University of Technology Thonburi, 2012

AWARDS

December 2013 **Recipient of an internship grant to work in Switzerland with CERN**, Summer 2014

November 2013 **Winner: Asia Pacific ICT Alliance Award 2013 (APICTA)** (Tertiary Student): *Hong Kong*

November 2013 **Winner: The International ICT Innovative Services Contest 2013**: *Taiwan*

August 2013 **Winner: Thailand ICT Award 2013 (TICTA)** (Tertiary Student): *Thailand*

July 2013 **Second runner-up: The Imagine Cup World Wide Final 2013** (Innovation Competition): *Russia*

April 2013 **National Winner: The Thailand Imagine Cup 2013**: *Thailand*

February 2013 **Winner: The 15th National Software Contest (NSC**) Science and Technology Category): *Thailand*

October 2012 **Recipient of the Thailand Research Fund – Master Research Grants**

June 2011 **Best Student Paper Award** (Title: Design and Development of Cloud-based Workflow Data Analytics Platform using RapidMiner): *Thai Grid and Cloud Conference 2011, Thailand*

June 2009 **Certificate for Academic Excellency** (Highest GPA in the class): *King Mongkut's University of Technology Thonburi*

May 2008 **Best Cheer Leading Team Award**, Engineering games 2008: *King Mongkut's University of Technology Thonburi*

PUBLICATIONS

2013 Pattanangkur, T., Tanupabrungson, S., Areekijseree, K. and Pumma, S., 2013, "The Design of SkyPACS: a High Performance Mobile Medical Imaging Solution", **The Symposium on GPU Computing and Applications 2013**, Singapore.

2012 Pumma, S., Achalakul, T. and Li Xiaorong, 2012, "Automatic VM Allocation for Scientific Application", **The 18th IEEE International Conference on Parallel and Distributed Systems (ICPADS) 2012**, pp.828-833, Singapore.

2012 Sukcharoen, P., Pumma, S., Mongkolsermporn, O., Achalakul, T. and Xiaorong Li, "Design and Analysis of a Cloud-based Epidemic Simulation Framework", 2012, **The 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON) 2012**, pp.1-4, Thailand.

2011 Pumma, S., Udomkasemsub, O., and Xiaorong Li, 2011, "Design and Development of Cloud-based Workflow Data Analytics Platform using RapidMiner", **The National Thai Grid and Cloud Conference 2011**, Thailand.

EXPERIENCES

July 2011 – 2013 **System analyst and developer**: *Innosoft*, KMUTT, Thailand

- Gather customer requirements and write the design specification
- Develop software
- Validate software

July 2011 –2013 **Research Assistant**: *Computer Engineering Dept*, KMUTT, Thailand

- Survey the existing technologies on cloud-based service platforms
- Design an open-service platform for Thai's SMEs

Summer 2011 **Research fellow**: *Institute of High Performance Computing, Agency for Science, Technology and Research (A*Star IHPC)*, Singapore

- Researched on effective workflow scheduling algorithm
- Designed and implemented the cloud-based workflow data analytics platform by using RapidMiner

November 2011 **Sale engineer (internship)**: *IBM Thailand*, Thailand

- Studied and presented the cloud computing technology and IBM cloud solution

SKILLS

Language Skills Thai (mother tongue), English (fluent, IELTS level 7.0)

Computer Skills

- Full command of Linux operating system (Ubuntu), shell script, and Xen Cloud Platform
- Excellence in C, C#, JAVA, SQL, VB.NET, HTML, and PHP programming