CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

RALB-HC: A Resource Aware Load Balancer for Heterogeneous Cluster

by

Usman Ahmed

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2018

I dedicate my dissertation to my family, teachers and friends. A special feeling of gratitude to my loving brother Mr Luqman Ahmed for his love, an extension of endless support and for providing encouragement.

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

CERTIFICATE OF APPROVAL

RALB-HC: A Resource Aware Load Balancer for Heterogenous Cluster

by

Usman Ahmed
MCS163021

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr Kashif Munir	FAST-NU, Islamabad.
(b)	Internal Examiner	Dr Masroor Ahmed	CUST, Islamabad.
(c)	Supervisor	Dr Muhammad Aleem	CUST, Islamabad.

Dr. Muhammad Aleem
Thesis Supervisor
September, 2018

Dr. Nayyer Masood
Head
Dept. of Computer Science
September, 2018

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
September, 2018

Author's Declaration

I, **Usman Ahmed (MCS163021)** of Computer Science Department hereby state that my MS thesis titled "**RALB-HC: A Resource Aware Load Balancer for Heterogenous Cluster**" is my own work and has not been submitted previously by me or by any other for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to cancel my MS Degree.

(Usman Ahmed)

Registration No: MCS163021

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "*RALB-HC: A Resource Aware Load Balancer for Heterogenous Cluster*" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above-titled thesis declare that no portion of my thesis has been plagiarized and any material used as a reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above-titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Usman Ahmed)

Registration No: MCS163021

List of Publications

It is certified that following publication/submission have been made out of the research work that has been carried out for this thesis:-

1. **Usman Ahmed**, Muhammad Aleem, Yasir Noman Khalid, Muhammad Arshad Islam and Muhammad Azhar Iqbal , "RALB-HC: A Resource Aware load balancer for heterogrous Cluster," *Concurrency and Computation, Practice and Experience (CCPE)* (submitted), 2018.

Acknowledgements

First and foremost, praises and thanks to **ALLAH (S.W.T)** for His showers of blessings throughout my research work to complete the research successfully.

I would like to express sincere gratitude to my research supervisor, **Dr. Muhammad Aleem** for giving me the opportunity to do research under their valuable guidance. Dr Muhammad Aleem way of encouragement and support is amicable. It was a great privilege and honour to work under his supervision. I would also thank **Parallel Computing Network (PCN)** research group for providing the motivational environment.

Finally, I would like to thank all my family. My friends who encourage and support me. This research would not have been possible without the valuable contribution of my brother **Mr. Luqman Ahmed**. His critics, support, guidance and advice helped me to complete my research.

Abstract

The homogeneous system consists of the similar type of multiple processors, whereas the heterogeneous system is equipped with a different type of multiple processes, i.e. Central Processing Units (CPUs) and Graphics Processing Units (GPUs). Heterogeneous systems based on CPUs and GPUs are becoming mainstream due to disparate processing and performance capabilities of these multi-core architectures. Mostly CPU is better suited to perform latency-sensitive tasks and incorporate architectural advances such as branch-prediction, out-of-order execution, and super-scalar capabilities. Whereas, many-core GPUs are more suited to perform data-parallel and throughput-sensitive tasks due to the inherent massive multi-threading capabilities. Despite much interest in heterogeneous systems, key scheduling challenges associated with them have not received much attention. Particularly, with highly shared resources having heterogeneous CPUGPU nodes, new application scheduling problems are arising. In such shared cluster environments, high utilization of resources and overall system throughput are important considerations in addition to the need for scaling a single application. In the heterogeneous computing environment (such as cloud), programmers map the applications only on CPUs or GPUs. However, the default process for device mapping is not able to produce ameliorate results, particularly on the heterogeneous cluster. If one resource of the cluster is powerful in terms of more computing capability, the scheduling schemes favour the powerful resource. In this scenario, the powerful resources are overloaded while all other resources are under-utilized. This load imbalance problem results in more energy consumption and increased execution time. In this research, a novel Resource-Aware scheduling for Heterogeneous Cluster (RALB-HC) is proposed that distributes workload based on resource computing capability and type of application. RALB-HC determine which data parallel application are like to best utilize a device. We show that device suitability is a good scheduling priority function and developed a data parallel application device suitability predictor. RALB-HC uses this prediction to prioritize and schedule tasks. The RALB-HC framework consists of two phases: (1) job mapping based on the availability of the resources and (2) the resource aware load balancing to

achieve higher resource utilization ratio. The experimental results on a large set of real-world as well as synthetic workloads show that the RALB-HC reduces execution time by 31.61%, increased resource utilization ratio by 67.8% and improved throughput 147.35% in comparison to baseline scheduling schemes.

Contents

List of Figures

List of Tables

Abbreviations

OpenCL	Open Computing Language
CPU	Central Processing Unit
GPU	Graphical Processing Unit
GPGPU	General Purpose Graphics Processing Unit
AMD	Advanced Micro Devices, a hardware company
ARUR	Average Resource Utilization Ratio
ROC	Reciever Optimization Curve
SVM	Support Vector Machine
ANN	Artificial Neural Network

Symbols

ET	Jobs execution time on each machine.
M_itask	Job assigned to i^{th} machine.
$ET - job_{M_i}$	Execution time of the jobs on the i^{th} machine.
$Total - ET$	Machines total execution time.
$Load_{M_i}$	Load of the i^{th}.
$Total - Load$	Cluster total load .
$Task - map$	Set of machines job mapping.
Con_{cric}	Convergence Criteria: Convergence value is repeated to the $\frac{numberofjobs}{2}$.
$ARUR_{Old}$	Average Resource Utilization Ratio Before Convergence Value Updation.
$ARUR_{New}$	Average Resource Utilization Ratio After Convergence Value Updation.
Id_{load}	Ideal load for the machine = $\frac{(totalload)}{numberofmachine}$.
$sel - job$	A job(having minimum execution time for the sel-machine) that migration needed.
$sel - machine$	A machine which is overloaded.
$mig - machine$	A machine which have load less than Id-load and minimum completion time for the job.
M_ijob	i^{th} machine number of jobs.

Chapter 1

Introduction

Today, most of the system is equipped with a multicore processor. Due to power consumption and transistor density constraints, the ever-increasing clock frequency trend is no longer possible [1, 27]. Therefore, multi-core architecture has been developed as a solution to the problem like power consumption, heat dissipation, and transistor density [27]. In multi-core architecture, multiple similar CPUs are integrated on the same integrated circuit. The system has the same type of processor are called *homogeneous system*. The hardware manufacturer increases computing power (in terms of parallelism) by increasing the number of the cores.

Nowadays, developers use parallel programming to speed up application [1, 2, 27]. An application is partitioned into parallel portions, each executing on a separate processor core. The parallel framework has further been strengthened up by the utilizing specialized processing unit having many-core such as a Graphical Processing Unit (GPU). GPUs was originally developed for handling graphical routines. However, in the current era, GPUs are being harnessed for general-purpose programming as well. GPU devices provide high-degree of parallelism. The huge computing potential of GPUs makes them suitable for data-parallel and throughput-oriented applications. GPUs are comprised of a large number of simple cores that can execute instructions in Single Instruction Multiple Data (SIMD) fashion. GPU processing, apart from its well-known 3D graphics rendering capabilities, can also perform mathematically intensive computations on very large

data sets. Whereas, the CPU consists of a small number of high-clocked cores that can provide fast response-time to a task and incorporates architectural advances such as branch-prediction, out-of-order execution, and super-scalar capabilities. CPUs can run the operating system and perform traditional serial tasks.

The Multi-core CPUs and many-core GPUs trend have initiated a new paradigm for computation processing called *heterogeneous computing*. Heterogeneous System Architecture (HSA) systems utilize multiple processor types (CPUs and GPUs). HSA are multi-core based systems that gain performance not just by adding cores, but also by incorporating specialized processing capabilities to handle complex task while being energy efficient. Due to immense data generation and huge processing power, the new application generating workloads with diverse requirements. The central processing unit (CPU) is unable to handle these diverse requirements. Heterogeneous computing, however, is designed to help and enable the efficient use of diverse processors like the CPU and GPU to efficiently handle these new emerging workloads. Intelligently utilizing the diverse processors help and enable new experiences while maximizing throughput and reducing turnaround time. Employing the diverse processors provides various opportunities to find at least the best of them that will truly excel at completing a particular workload. Some processors are rather inefficient at certain jobs while excelling in others. Once we realize that each type of processor has its own strength, we can opportunistically and intelligently choose the appropriate one for the specific workload. With help of heterogeneous computing, different processors can be designed to work together enabling new user experiences.

1.1 OpenCL: Heterogenous Programming Framework

Open Compute Language (OpenCL) [1] emerged as an industry standard to develop data-parallel applications for heterogeneous multi and many acore architectures. Major vendors in the computer hardware industry such as Intel , AMD,

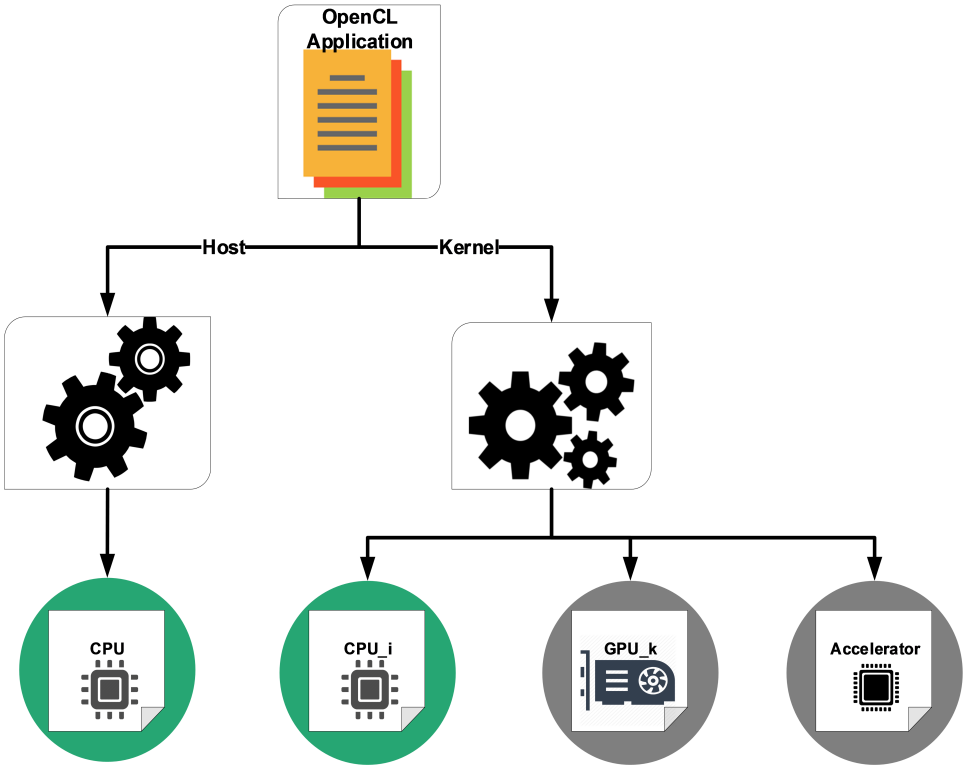

FIGURE 1.1: OpenCL Task Execution

and NVIDIA etc. support OpenCL platform [1]. The OpenCL execution model is depicted in Figure 1 where a CPU executes the serial portion of an OpenCL program (known as the host) whereas the parallel portion (known as the kernel) can be executed seamlessly on a CPU, a GPU, or any other supported accelerator device. In spite of the fact that it gives convenient usefulness, its execution will definitely fluctuate over distinctive parts of the heterogeneous framework. The reason is that, while one application may have low execution time on the GPU than on the CPU, another application's performance might degrade significantly if assigned to the GPU while attaining low execution time on the CPU. Programmers normally assign tasks to either a CPU or GPU, due to which another processing-unit remains idle. For example, if tasks are assigned to a GPU device, this leaves the CPU idle waiting for the scheduled tasks to complete as shown in Figure 1.1. In this research, we will be using OpenCL because of its portability and a large number of supported compute-devices.

1.2 Application Scheduling on Heterogeneous Machines

Scheduling in heterogeneous machines has been discussed in many research articles and several solutions have been suggested [2–5]. The scheduler decides a particular data parallel application should be assigned to which device, i.e., a CPU or to a GPU. The proposed schedules are suitable when the amount of work to be performed or data to be processed is known a prior [4–6]. These scheduling schemes carry low scheduling-overhead, but they are not always provided optimal task partitioning. Some researchers perform task mapping to a compute-unit at runtime. The main advantage of runtime task scheduling is that the decision to map the task is more optimal (considering the runtime attributes of the application and machine) [7–11]. Scheduling decisions can be adjusted during execution of a program. Major cons of run time scheduling are the increased complexity and higher scheduling overhead.

Supervised machine learning model has also been used and proven to be effective in learning to optimize scheduling[3, 61]. Code features are used to characterize an application[4, 12]. The code features include the number of instructions, and parallel runtime parameters, such as the number of work items. At compile time, application abstract syntax tree is generated by using a compiler named as CLang and LLVM [65]. The abstract syntax tree gives information about application behaviour, i.e. the number and type of operation used in the application, the count of barrier occurs, the number of blocks within the application, the count for the load operation performed by the application and the count of store operations performed by the application [27]. The count of each code feature (number and type of operation, barrier, blocks, load/store operation) in an application is used as features values. The features in the feature vector are classified into two types, i.e. Static features and Dynamic features. Static code-features such as the number of int operations, local memory access percentage are extracted at compile time while dynamic features extracted includes input workload. All feature values combined to form a feature vector. These values are then used as input to a predictive model

that is based on machine learning. The predictor is trained on the extracted feature vectors. The features are selected on the basis of their contribution to predicting the output. The motivation behind using a reduced feature set (for predictive models) is the less redundant data, reduced overfitting issues, improved accuracy, and decreased training time of the algorithm.

1.3 Heterogeneous Scheduling Issues

In the heterogeneous computing environment, programmers usually map application only on CPUs or GPUs [3, 4]. The programmer uses a default scheduling scheme where a parallel portion of a program is assigned to the GPU while CPU executes serial portion (kernel management) of a program [12, 13]. The CPU remains idle while all the computation is performed by the GPU, although by using OpenCL, we can execute a program on both CPU and GPU [14]. This is the wastage of CPU resources in terms of power and energy consumption and not performing any useful task [14–16]. To address wastage of resource issue, some researchers have proposed scheduling mechanisms, however, most of them handle a specific type of application or perform on a single data parallel application scheduling [3]. Moreover, the number of techniques depends upon code splitting overhead and required profiling of applications [3, 4, 12–14]. Some authors have proposed frameworks for the utilization of multiple heterogeneous machines, however, these do not resolve the load imbalance problem [15–17]. The vast majority of the literature evaluated scheduling tasks in the heterogeneous environment, however, these techniques have the following issues:

1. Most of them required a data parallel application code splitting overhead to split tasks among CPU & GPU device. This data parallel application code splitting will result in additional time overhead.

2. Existing solution proposed a profiling-based scheduling method. They used code instrumentation and profiled time to scheduled application to a device i.e. CPU or GPU. This profiling required time overhead.

3. To the best of our knowledge, no prior work attempts to do a load balanced task scheduling in a cluster of the heterogeneous system using a machine learning approach.

1.4 Problem Statement

In a heterogeneous environment, the vast majority of the literature evaluated scheduling tasks by using the data parallel application code approach or profiling heuristics. To the best of our knowledge, none of the schemes has focused on load balancing task scheduling in a cluster of the heterogeneous system by considering data parallel application device suitability and its execution time.

1.5 Research Questions

The critical analysis of the literature survey has led us to the following research gaps, which have been focused in this thesis:

1. RQ-1: How to design and develop a load balancing scheduling algorithm to achieve the minimal execution time, maximal throughput and improved resource utilization?

2. RQ-2: To analyse optimization technique for design device suitability classifier and execution time predictor?

    2.1. RQ-2.1: Which set of features plays an important role to predict data parallel application device suitability?

    2.2. RQ-2.2: Which are the most important factors, forecasting the execution time of data parallel application?

1.6 Purpose

Our goal is to accelerate a set of applications using the aggregate set of resources in the cluster of heterogeneous machines. The acceleration is achieved by considering the distribution of workload, to minimize the completion time of the job pool and to increase the resource utilization.

1.7 Scope

In this work, a novel Resource-Aware scheduling for Heterogeneous Cluster (RALB-HC) is proposed. The device prioritized list and resource performance estimation are achieved using two machine learning based predictive models, i.e. device suitability predictor and application time estimator [4, 12, 27, 65]. The device suitability predictive model, predicts the prioritized list of resources that are highly suitable for a given application in a job pool. Moreover, application performance on the prioritized resource has also been forecast. The device prioritized list and resource performance estimation are then used by a proposed resource aware load balancer for the heterogeneous cluster (RALB-HC). The results of this study will be immensely valuable to increase cluster of heterogeneous system utilization as well as for overall system throughput. ‘

1.8 Application

The research will assist the user to improve the cluster performance in terms of execution time, throughput and resource utilization.

1.9 Dissertation Organization

The rest of the document is divided into the following sections. In Chapter 2, we present a review of state-of-the-art approaches and a critical analysis of these research techniques. Chapter 3, outlines the methodology of the study, the proposed system architecture and performance evaluation metrics are discussed. Chapter 4 encompasses the details regarding implementation of the proposed RALB-HC with the detail of experimentation performed on a large set of application. Each experiment is explained in detail and evaluation is performed comprehensively. The Chapter 5 concludes the research work and suggested the future directions.

Chapter 2

Literature Review

Task scheduling is a non-trivial problem that requires optimal mapping of tasks to the processor so that the overall execution time of applications is reduced. The scheduling decision becomes more difficult when we have a heterogeneous cluster in which each compute unit has diversified set of characteristics. Heterogeneous systems based on Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are becoming mainstream due to disparate processing and performance capabilities of these multi-core architectures. Mostly, CPUs are better suited to perform latency sensitive tasks and incorporate architectural advances such as branch-prediction, out-of-order execution, and super-scalar capabilities [46]. Whereas, many-core GPUs are more suited to perform data-parallel and throughput sensitive tasks [47] due to the inherent massive multi-threading capabilities [47]. The CPU has a limited number of powerful and complex cores that are generalized to execute different types of applications efficiently, whereas GPU contains a large number of simplified cores that are mainly specialized to execute data-parallel portions of the program. Therefore, while scheduling heterogeneity of computing devices should also be considered to effectively map computation to processors. Programmers normally assign tasks to either a CPU or GPU due to which other processing-unit remains idle e.g. if tasks are assigned to a GPU device this leaves the CPU idle just waiting for the scheduled tasks to complete. Various researchers have proposed scheduling techniques for heterogeneous platforms

[5, 18–21]. Some of them have split a data parallel application between the CPU and GPU while others have improved the throughput and resource utilization by scheduling pool of applications. The machine learning based predictive modelling is considered to be a powerful method for optimizing parallel programs [4, 12–14, 22]. The predictive model is trained to learn from its set of examples and have an adaptive behaviour for varying platforms. By using the scheduling technique [4, 12–14, 22], severe load imbalance is introduced between CPU_i and GPU_k due to CPU_i only managing execution of kernel on GPU_k and taking no part in actual computation. The idle time that CPU_i spent while waiting for GPU_k to complete kernels execution is not desirable. Ideally, a schedule is required that can schedule data parallel application to both CPU and GPU in such a way that all processors in the cluster can complete processing at the same time. In this way energy consumption and heat dissipation due to idling processor are reduced but, more importantly, the execution time of Job Pool will also be reduced significantly. The optimal device selection is a key for any scheduler schemes in a heterogeneous environment.

Perez et al[20] presented a Maat library that performs load balancing of single kernel across heterogeneous devices [20]. According to Perez et al, the programmer does not utilize heterogeneity in the optimal way as they consider CPU device for sequential tasks whereas GPU is considered for the parallel task. This inflexible approach results in wastage of computing power [20]. With Maat, the programmer just needs to build up a parallel version of the kernel program, which selects a load balancing method and runs it on the all the available resources. In Perez et al [20] approach, there is no need for extra programming effort, as same raw kernel code is utilized. Moreover, at runtime, the predictive model is able to determine device suitability, as well as application time estimation, is made to achieve maximal throughput.

Luke et al [5] have also addressed the problem of optimizing the utilization of available resources. They have focused on the need for automated mapping of processing elements to the available resources [5]. According to Luk et al[5], programmer

utilization of heterogeneous platform is able to adapt according to hardware/-software configuration. Therefore, a system name Qilin is proposed that utilize machine learning to classify kernel code. The kernel code is partitioned into the CPU and GPU device. The Qilin index the execution time of an application in a database. The recorded information is then utilized by the Qilin to project execution time of new arrived application and to schedule it accordingly. Whenever the hardware configuration changes, Qilin initiates a new training session. The Qilin requires offline profiling and code partition overhead whereas RALB-HC method does not require these overheads.

Huchant et al [19] have proposed an automatic runtime technique that schedules OpenCL kernel code across Heterogeneous devices. Huchant et al[19] technique is able to solve issues causing from the heterogeneity i.e. Communications, load-balancing and issues caused by the iterative computation. The technique is divided into two main approaches, i.e. Static and dynamic. In static phase, kernel code is transformed into partition ready kernels, which are then mapped into different devices. The execution time of mapped kernel is noted and then in a dynamic phase, queuing of the partitioned kernel is adjusted to achieve optimized throughput. This technique differs from our technique as it mapped a single OpenCL kernel, whereas, our technique manages to schedule pool of OpenCL applications.

Albayrak et al [23] have addressed the need for optimal mapping among different heterogeneous devices, CPU or GPU. According to them, in the multi-application environment, different kernels exhibit different characteristics. Some of them run faster on the GPU, others may refer to execute on CPU due to data transfer cost. However, there is a need to map the kernel to the proper device to improve the overall performance of an application. Albayrak et al [23] have proposed a profiling-based scheduling method to map OpenCL application [23]. The data dependencies and execution time are profiled. Then, by the use of a greedy algorithm, the kernel is scheduled to device i.e. CPU or GPU. The proposed algorithm is able to achieve the optimal result for scheduling multiple kernels of single application only. However, RALB-HC scheme does not require offline profiling overhead. RALB-HC is able to schedule single and multi-kernel application within a

batch of job pool.

In *A Framework for OpenCL Task Scheduling on Heterogeneous Multicores* Augonent et al [11] explain the need of execution model that unified the computation unit on heterogeneous architecture. Augonent et al [11] have proposed StarPU, a runtime system that provides an execution environment for numerical kernels. The system focuses on four policies i.e. priority, non-priority base, Ws policy, w-rand policy and heft-tm policy. In the priority-based scheme, the task with higher priority is given preference. Whereas in the no-priority scheme, priority is not considered while assigning the task to a processor. In Ws policy, the task assignment is based on the principle of work stealing. The above-mentioned policies follow Greedy approach i.e. as soon as the processor becomes available, a task is assigned to it. In w-rand policy, an acceleration factor is assigned to each processing unit. Afterwards, the task is assigned to that processing unit having the probability proportional to the acceleration factor ratio. In heft-tm policy, assessment of past performance of processing unit is considered. On the basis of that assessment, the task is assigned to that processing unit which provides optimized execution time. Augonent et al. [11] have proposed a model that considers mathematical application i.e. Matrix Multiplication and LU decomposition whereas proposed scheduling is capable to schedule diverse application in multi- GPU configuration.

Becchi et al [18] have proposed a solution for the optimization of the heterogeneous environment with minimum knowledge of the underlying architecture details. The proposed schedule by Becchi et al [18] considers the execution history as well as data transfer overheads and then schedules the kernel function call on either a CPU or a GPU. This technique requires the implementation of both the CPU and GPU version of the kernel. The approach has an edge over other scheduling heuristics because it attempts to bring computational data near to processing unit by considering profiled execution time. Becchi et als [18] scheme requires an offline profiling overhead whereas, proposed RALB-HC scheduler is capable of scheduling number of data parallel applications without profiling.

In a similar study titled *A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures*, Belviranli et al examine the issues in resource utilization of heterogeneous environment and proposed a scheduling mechanism named as HDSS [7]. It partitions the workload among processing units, i.e. CPU and GPU. This results in improvement of kernel execution time. HDSS has two phases, i.e. profiling phase and adaptive phase. The computation power of each processing unit is evaluated by assigning the some number of loop operations in the profiling phase. While remaining loop operation is assigned based on the processing speed in the adaptive phase. Both phases help in balancing the load on heterogeneous computing devices. The proposed RAlB-HC is not dependent on job splitting and any kind of raw code transformation.

According to Binotto et al[24], distribution of workload among processing units plays a key role in the heterogeneous environment. Binotto et al. [24] have addressed that the cost of the task assigned to the processing is non-deterministic and that cost can be affected by parameters not known a priori. Therefore, Binotto et al.[24] have proposed a system which assigns data parallel tasks to CPU or GPU. The kernel code is split among several tasks. After that, execution time and performance are indexed by an online profile in a database. Whenever a new task arrives for execution, sorted performance profile determines the scheduling of task to either a CPU or GPU device. The approach of Binotto et al [24] requires online profiling as well as kernel code transformation.

In *An (ir)regularity-aware task scheduler for heterogeneous platforms* Gregg et al. [25] examine the processing unit performance and address that unit performance are reduced due to work required for offline training as well as generating partitioning of code [25]. As a result, Boyer et al proposed a dynamic approach to partition workload among processing unit without offline training [25]. The algorithm divides the workload into the number of chunks and then schedules those chunks to either CPU or GPU. On the basis of previous execution of chunks, the sizes of chunks are increased or decreased exponentially. The load is balanced by scheduling larger size chunks to the fast processing unit and smaller size chunk to the slowest unit. Kaleem et al. have also used the splitting approach based

on profiling [26]. However, RALB-HC ensures lower execution time by achieving maximal resource utilization ratio and maximal throughput.

Heterogeneous computing system gets improved performance by utilizing the powerful CPU as well as the GPU. According to Choi et al [8], the device selection is the most critical factors in determining the performance of application [8]. Therefore, Choi et al.[8] have estimated the execution time, which determines the schedule of the application on a CPU or a GPU device. The model requires execution history of application to train and map newly arrived applications to that device, which has finished job earlier. The total execution time of the application (on that device) and the execution time of the currently executing application are used to estimate finish time of an application. In contrast, our proposed model dynamically determines not only the device suitability but also device relative speed.

Heterogeneous multi-core platform has superior performance over the homogeneous system. However, according to Grewe et al [4], the optimized result can only be achieved if the task is accurately mapped to the appropriate processing unit. Grewe et al [4] have developed a portable partitioning scheme for the OpenCL program. At compile time, the model extracts static code features, i.e. int operations, float operations, barriers, work items etc. [4]. Then pre-trained model SVM is utilized to predict whether to map a kernel to a CPU, GPU or to partition the kernel among available computing devices. Grewe et al [4] *GPU only model* achieve 91% accuracy, whereas the *CPU only model* achieved 95% accuracy. In contrast, the proposed RALB-HC scheduling model not only predicts device suitability but also predicts relative speedup gains over the non-selected device. Moreover, the Grewe et als model schedules a single OpenCL kernel across heterogeneous devices, whereas our approach schedules a job pool of OpenCL applications.

In *A Framework for OpenCL Task Scheduling on Heterogeneous Multicores* [3] proposed a predictive model, which is the extension of the work proposed by the Grewe et al [3, 4] conducted an in-depth analysis of the control flow divergence and

its impact on the program partition. The predictive model (Ghose et al., 2016) [3, 4] has included branch divergence in the code feature. Ghose et al used two classification models i.e. decision tree and radial base network. The classifier is trained on three types of model i.e. CPU-GPU inclusive, Partition CPU-GPU and Combined model. The cross-validation method is deployed to assess the effectiveness of newly proposed features. The CPU-GPU inclusive achieved 89% accuracy and the Partition CPU-GPU model achieved 80.84% and 81.23 % respectively. In contrast to their proposed schedule, our technique ensures multi-node scheduling of tasks among the processing unit and can achieve high throughput.

According to [13] perspective, full utilization of heterogeneous environment is a challenging task due to the difference in the processing capabilities, memory availability and communication latencies of different computational resources. Therefore Kofler et al [13] proposed an ANN based predictive model. The basic task of a predictive model is to dynamically partition the given task on a CPU and a GPU. Kofler et al [13] used Insieme source to source compiler to translate a kernel code into multi-device kernel code. The dynamic partition is based on the Artificial Neural Network (ANN) predictive model. The feature set includes static code features and dynamic input sensitive features (e.g. Data-transfer size of the split-able buffer). The partitioning task is further to improve from 2% to 7% by using Principal Component Analysis. The test set achieved 87.5 per cent results. Kofler et al[13] have partitioned the program and achieved high accuracy. Our proposed scheduler selects an optimal device as well as do a scheduled task on a cluster of devices by using the application device performance of the selected device. Moreover, the proposed schedule does not require kernel splitting.

Wen et al [27] addressed the need for optimal utilization of heterogeneous environment. According to Wen et al [27], in order to get increased system throughput and decrease turnout time, there is a need to determine when and where to map different applications [27]. The feature set of the predictive model includes a static code feature (number of instructions, load/store operations etc.) and dynamic code features (input size, output size and global work size etc.). Wen et al. have converted a regression problem into classifications by labelling the kernel

as high-speedup if the measured GPU-speedup is larger than 4, otherwise, it will be labelled as low-speedup [27]. The high-speedup is mapped to the GPU device, whereas low-speedup is mapped to a CPU device. The support vector machine (radial base kernel) is used as the classification model. In this research, the scheduler assigns OpenCL kernels by considering the requirement of the application and the device's computing capabilities.

In *Adaptive optimization for OpenCL programs on embedded heterogeneous systems* Wen et al [14] address that certain application performance is maximized when assigned to the single computing device and sometimes sharing among computing device results in improved performance [14]. Therefore, Wen et al proposed a predictive model that determines whether an application kernel needed to be scheduled to a single device or it should be combined with other kernels to improve execution performance of a job pool [14]. The feature set of the predictive model includes static and dynamic features, which further extend to separate kernel features and concurrent kernel feature. The decision tree classification algorithm is used to classify the kernel to a suitable device. The model is trained using separate kernel features and concurrent kernel feature. The first model separates the kernel into the CPU and GPU, which is based on estimated device affinity. The second model determines whether or not to merge two GPU kernels. In contrast, our proposed scheduler maps the devices on a cluster of CPUs or GPUs, by using device suitability predictor as well as application performance estimation.

2.1 Task Scheduling Heuristics

In a cluster, if one resource of the cluster is very powerful then the scheduling schemes favour to the powerful resource. The powerful resource will be overloaded and all other resources will be under-utilized. This load imbalance problem results in a more energy consumption [35, 66]. Several scheduling mechanisms have this

problem which results in underutilization of Cloud resources[35]. These scheduling mechanisms include Minimum Completion time (MCT), Min-Min, Resource-Aware Scheduling Algorithm (RASA), Max-Min and Task-Aware Scheduling Algorithm (TASA) [35, 49–52].

Minimum Completion Time (MCT) scheduling mechanism assigns the job to that resource which has a minimum completion time. The current load on the resource is used to identify an appropriate resource for the scheduling the job [35, 49–52]. At each step, MCT needs to find the resource which has the least execution for a given job. The search to find minimum time machine result in scheduling overhead. After experimentation, it is found that the MCT performs better as compared to the random selection and round-robin scheduling schemes. However, MCT assigns more jobs to the powerful resource due to which load imbalance problem occurs [55].

Min-Min scheduling heuristic follows the MCT mechanism. Min-Min scheduling consists of two steps. In the first step, resources finish time is determined for all candidate jobs [35, 49, 52]. The second step is performed by assigning the selected job to the machine which has the minimum finish time. After each job mapping decision, the ready time of each resource gets updated. In this way, all the batch of the job is mapped among machines [53, 54]. Min-Min mechanism favours smaller jobs which result in low resource utilization for a job pool.

The max-min mechanism is similar to the Min-Min scheme. However, the selection policy of both scheduling mechanism is different but their functionality is almost identical [35, 56]. Like Min-Min, Max-Min also consists of two steps. The first step is identical to the Min-Min, resources finish time is determined for all candidate jobs. In the second step, the resource is assigned to that job which has maximum earliest finish time for it. The resource ready time is updated on each job scheduling decision [57, 58]. Max-Min mechanism favours larger jobs which result in low resource utilization for a job pool and cause load imbalance.

A grid computing load balancer named Resource Aware Scheduling Algorithm (RASA) alternatively uses the Max-Min and Min-Min [35]. Firstly, resources finish

time is determined for all candidate jobs. The mapping is done by considering the number of jobs, if a number of jobs are odd then Min-Min heuristic is employed first, an otherwise Max-Min heuristic is used [58, 59]. After that, Min-Min and Max-Min are alternatively used to schedule the remaining jobs. RASA provide fair scheduling mechanism for large and small size jobs. However, the load imbalance problem still exists if the job pool contains a larger number of big jobs.

The Sufferage mechanism calculates the sufferage value for all jobs. The sufferage value is the difference between its MCT and the second MCT for each job in the pool [35, 54, 55]. The mapping is done by selecting the job which has the largest sufferage value and then mapping it on the machine having minimum execution time for it. The Sufferage mechanism results in minimum makespan, however, the calculation overhead of sufferage value is very large in each job mapping [60, 62, 63]. In the majority of the cases, the sufferage mechanism performs better than the Max-Min, MCT and Min-Min.

Task aware scheduling algorithm (TASA) follows the same RASA type alternative mechanism [34, 35]. However, it uses the Min-Min and Sufferage heuristics. Firstly, the TASA uses the Min-Min mechanism to map jobs. After that, the Sufferage value is calculated to map job among the resource [34]. TASA generates better makespan and resource utilization as compared to Sufferage, RASA, Min-Min and Max-Min.

A Resource-Aware Load Balancing Algorithm (RALBA) distributes load based on the computation capability of the resource. The RALBA comprises a two-phased i.e. fill scheduler and spill scheduler. The fill scheduler loads the batch of the job, according to the capacity of the resource. If the all the machine capacity is full then spill scheduler map all the job by using the minimum completion time. RALBA has achieved significant improvement in execution time resource utilization and throughput against above-mentioned scheduling heuristics, however, However, RALBA does not support SLA-aware scheduling of jobs.

TABLE 2.1: Summary of state-of-the-art Task scheduling Heuristics

Heuristic	Makespan Improvement	Favours Smaller jobs	Favours Larger jobs	Load Imbalance	Weakness
MCT	✓	✗	✗	✓	Fasters resources allocation with number of jobs.
Min-Min	✓	✓	✗	✓	Favors smaller jobs and penalizes the larger jobs.
Max-Min	✓	✗	✓	✓	Load imbalance when larger jobs are in the job pool.
Sufferage	✓	✗	✗	✓	Computing Sufferage value overhead in each job mapping.
TASA	✓	✓	✗	✓	TASA doesnt address load balancing.
RASA	✓	✗	✓	✓	Penalizes smaller jobs in some cases.
RALBA	✓	✗	✗	✓	Penalizes smaller jobs in some cases.

TABLE 2.2: Critical review of state-of-the-art heterogeneous scheduling approaches.

Ref	Methodology	Weakness	Strengths
[14]	Wen et al used a static and dynamic code feature. Predictive modelling (to predict whether an application kernel needed to be scheduled to a single device or it should be combined with other kernels.	The limited number of application kernel is used. The methodology does not incorporate multi-node application. The methodology does not incorporate multi-node application. Doesnt incorporate load balancing.	The feature set has both static and dynamic code features that will able to improve the accuracy of the predictive model.
[22]	Taylor et al used static and dynamic feature to predict whether an application kernel needs to be scheduled on mobile computing CPU or GPU.	The methodology does not incorporate the load imbalance problem.	The static and dynamic feature will increase predicted model performance
[3]	Ghose et al proposed control divergence features in addition to static and dynamic kernel features. The predictive model predicts that whether an application kernel should run on CPU or a GPU or kernel should be split among the CPU and GPU i.e. Partition CPU-GPU.	Methodology required kernel splitting overhead to split tasks among the CPU and GPU device. The methodology, not able incorporates load imbalance problem.	The control flow divergence analysis increases the accuracy of predictive models.
[27]	Wen et al used a static and dynamic code feature. Predictive model used to forecast whether an application needs to run on a CPU or a GPU.	Wen et al have not addressed the load imbalance problem and device suitability among Multi-node devices.	Static and dynamic kernel code features will able to improve, enhance the performance of predictor.
[13]	Kofler et al uses static, dynamic and inputs sensitive kernel features. An OpenCL application Kernel is portioned into two tasks that are scheduled on a CPU and GPU.	The proposed strategy required kernel splitting among the CPU and GPU device, which results in additional time overhead. Moreover, it does not incorporate a multi-node application splitting strategy.	Input-sensitive features will help to enhance predictive models.
[4]	Grewe et al develop a machine-learning based compiler model that accurately predicts the best partitioning of a task given only static code features.	Although the kernel splitting technique is automatic but still consumes additional time overhead. Multi-node application scheduling is not achievable.	The static features have achieved 3.03 overall on CPU strategy and 1.55 over an all on GPU strategy.

2.2 Critical Analysis

After the comprehensive analysis of state-of-the-art approaches, we found techniques for heterogeneous scheduling on a heterogeneous machine. The majority of heterogeneous scheduling schemes do not address the problem of overloading, which results in longer execution time and low resource utilization [4, 13, 14, 22, 27, 34, 35, 49–60, 62, 63]. There are several techniques that do not consider device suitability and this often results in low resource utilization [5, 18–21]. A few techniques use the machine learning approach to predict the suitable device and then split the kernel code among the CPU and GPU [4, 12–14, 22]. The kernel splitting overhead required additional time overhead. The number of research does not consider multi-node application splitting or merger technique and load balancing (if a large number of kernel always suits one device then, overall throughput trends to decrease) issue in the cluster of a heterogeneous environment. The analysis of scheduling heuristics (our experiments) revealed that having minimal makespan and high throughput cannot guarantee the load balanced scheduling.

Chapter 3

Methodology

In the heterogeneous computing environment, researchers tend to map applications on CPUs or GPUs only. However, this decision making is not considered as effective when there is a multi-node or cluster of the heterogeneous system [67]. Moreover, the scheduler receives the number of jobs and then the scheduler makes the mapping decision to place an application of the available computing devices. That decision about the work distribution among a set of resources should be balanced to achieve the maximal throughput. Load balanced scheduling is a critical issue in the heterogeneous system and it has become even more crucial for the cluster of heterogeneous devices (because of multi-level of parallelism) [66, 67]. It is very difficult for a researchers to decide the mapping of jobs to a variety of heterogeneous computing devices [66]. Therefore, an automated strategy should be devised for the efficient scheduling of the jobs. Programmers normally use the default scheduling strategy. In this type of scheduling, parallel portion of a program (kernel) is assigned to the GPU while CPU executes serial portion (kernel management) of a program. The CPU remains idle while all the computation is performed by the GPU. The idle time results in wastage of precious CPU resources which are consuming power. It is difficult for the programmer to know the nature of each job and select device according to the requirement of the job as well as hardware properties. Application mapping according to the suited devices is a

difficult task. To address this problem, we propose a machine learning based predictive model that predicts application device suitability as well as its expected performances. Device suitability prediction is utilized by the scheduler for the optimal mapping of workload. The resource aware load balancer for the heterogeneous cluster (RALBHC) is proposed that utilizes the predicted model output and then balances the load among the available resource. In order to evaluate the performance of the proposed technique, the detailed methodology is presented in Figure 3.1.

A user or multiple users submit a data parallel application and the proposed RALB-HC scheduling system assess submitted job by extracting static code features. The RALB-HC scheduling consist code feature extractor, device suitability classifier, application estimator, and scheduling heuristic.

The feature extraction mechanism is explained in the section 4.3.2. Extracted code features along with the input data size of a job are then provided to the Device Suitability Classifier module, which classifies the submitted jobs, according to device suitability (a potential best-performing device for that job i.e., a CPU_i or a GPU_k). The application time estimator predicts the execution time of the job on all the available resources. The output of both prediction models is then used by the resource-aware. The next section presents the detailed discussion of the Device Suitability Classifier and the Application estimator modules.

3.1 Resource-Aware Load Balancer for Heterogeneous Cluster (RALB-HC): System Architecture

The proposed algorithm system architecture is shown in Figure 3.2. The batch of jobs is provided to the RALB-HC and load balancing of application is performed. It consists of two sub schedulers i.e. resource selector and load balancer. Resource selector first computes a load of each machine followed machine selection having

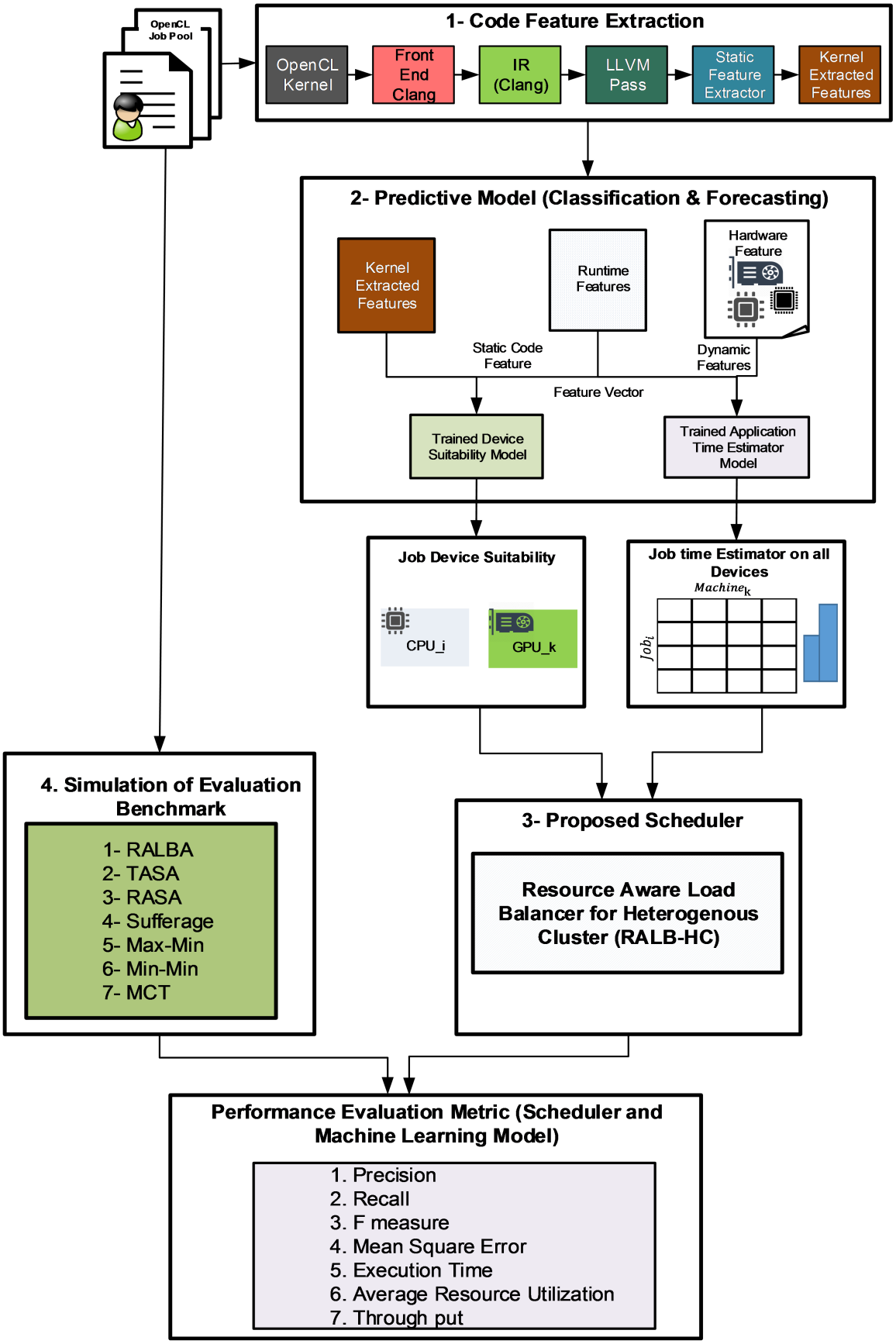

FIGURE 3.1: Schematic diagram representing proposed RALB-HC Methodology

a maximum load on it. After that, it selects that job which has the minimum completion time for the selected machine. Afterwards, it selects the migration machine, which has load less than *(total load/number of machines)* and the minimum completion time for the given job. If the machine is not available, then the machine that has 2nd minimum completion time for the job will be selected and the loop continues until the job assigned to any machine. After that, the job is removed from the selected machine and mapped to the newly selected machine. In this way, all the jobs on the selected machine are mapped to new machines until the selected machines load is less than the total load/number of machines. This task is performed by the load balancer. After that, the selected machine is removed from the list and resource selector selects the machine which has the maximum load. This process continues until all machine load is distributed among available resources. After each iteration, resource utilization is calculated and compared. If there is an improvement in the resource utilization then the new resource utilization value becomes the old resource utilization value and machine mapping is saved. After saving the machine mapping, the convergence value is set to zero. If there is no improvement, then the convergence value is incremented. Based on experimentation value, we have selected the algorithm converges value that should be less than half of the total number of jobs. The convergence indicated that maximum resource utilization is achieved.

The proposed algorithm RALB-HC (shown in Algorithm 1) has two primary modules, i.e. resource selector (Algorithm 2) and load balancer (Algorithm 3). Symbol table explains the interpretations of variables used in Algorithm 1-3.

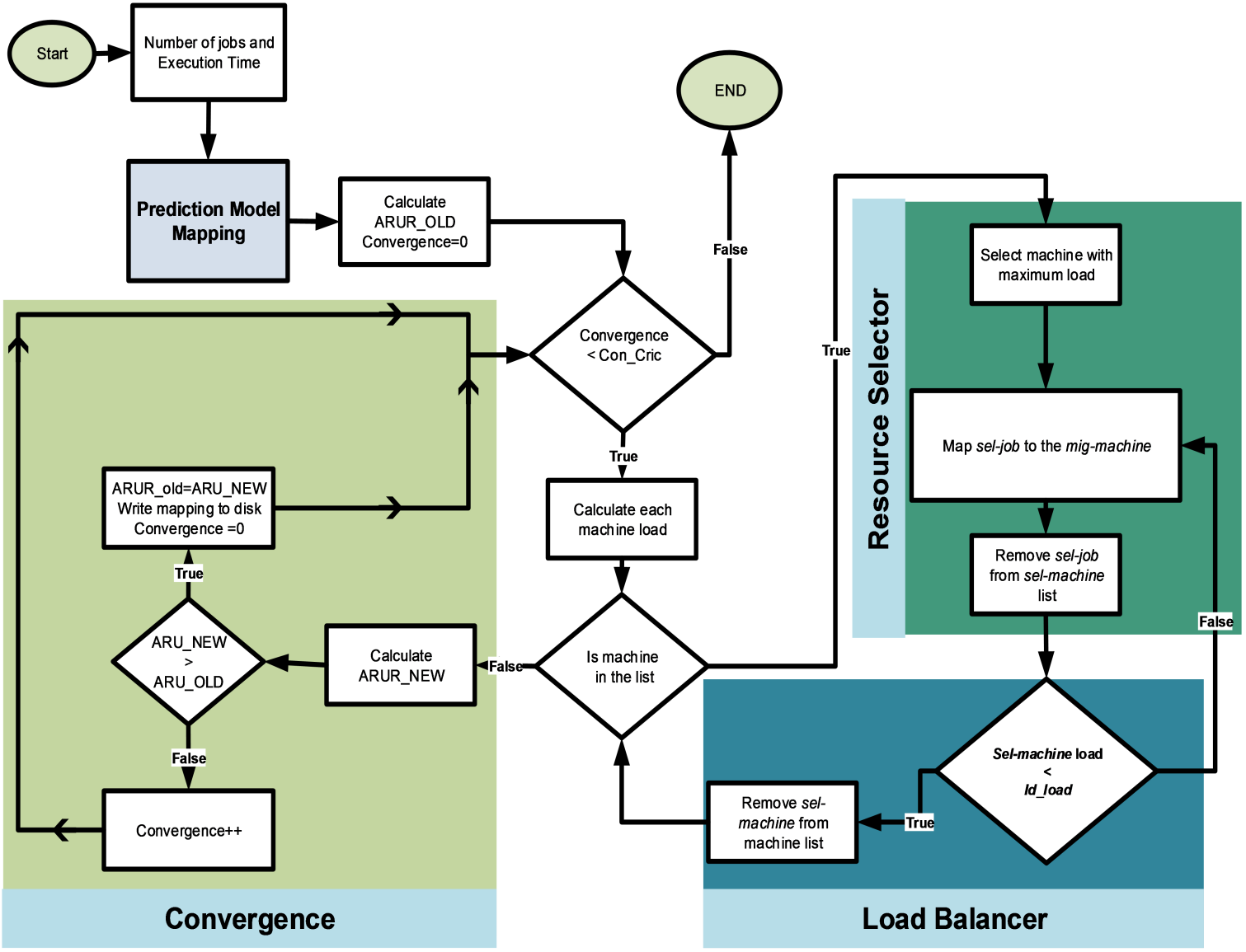

FIGURE 3.2: RALB-HC System Architecture

3.1.1 RALB-HC (Algorithm 1) and $ARUR_{OLD}$ Calculation

For the Algorithm 1, the input is the execution time of all jobs on every machine and output is the load balanced mapping of jobs on all available machines. Firstly, the jobs are mapped to the machine that has the minimum execution time for each job in case of synthetic and Google-like workload. In case of realistic load, the classification algorithm mapping will be utilized *(line 1 of Algorithm 1)*. The convergence value is set to zero *(line 2 of Algorithm 1)*. This convergence value is used for optimization explained in the previous section. The average resource utilization is calculated *(line 3 of Algorithm 1)* then, while loop starts with the condition if the convergence is less than the half of the total jobs the loop continue else its stops *(line 4 of algorithm 1)*. By taking the execution time of all jobs on each machine and current mapping of each machine, the resource selector algorithm 2 returns improved task mapping *(line 5 of algorithm 1)*. Afterwards, once again the average resource utilization is calculated (line 6 of algorithm 1). If the new average resource utilization is greater than old average resource utilization *(line 7 of algorithm 1)*, the convergence value is set to zero *(line 8 of algorithm 1)*, average resource utilization new becomes average resource utilization old (line 9 of algorithm 1) and the mapping of task is saved *(line 10 of algorithm 1)*. If the there is no improvement in the average resource utilization then convergence is incremented *(lines 11-13, Algorithm 1)*.

3.2 RALB-HC Resource Selector (Algorithm 2)

For the Algorithm 2, the input is the execution of all jobs on each machine, and task mapping of all machines. The output is the load balanced task mapping. The algorithm calculates the sum of execution time for all jobs in each machine *(lined 1-2 of Algorithm* 2). The total execution time is calculated by summing the execution time of each machine *(line 3 of algorithm 2)*. A load of each machine is calculated by dividing the machine execution time to total execution time *(lines 4-6 of Algorithm 2)*. The while loop starts from zero to the number of the machines

Algorithm 1 : RALB-HC: Proposed Scheduler for Heterogeneous Cluster

Input: Execution time of all jobs on every machine.
Output: Load Balanced mapping of jobs on machines

1: **Select** $M_i \leftarrow Predictionmodel()$
2: $convergence \leftarrow 0$
3: $ARUR_{old} \leftarrow \frac{mean(ready_{time} - M_i)}{makespan}$
4: **while** $convergence < Con_{cric}$ **do**
5: $task_{map}[\,] \leftarrow resource_{selector}(ex_{time}[M_1, ..., M_n], task_{map}[M1_{task}, ...Mn_{task}])$
6: $ARUR_{new} \leftarrow \frac{mean(ready_{time} - M_i)}{makespan}$
7: **if** $ARUR_{new} > ARUR_{old}$ **then**
8: $convergence \leftarrow 0$
9: $ARUR_{old} \leftarrow ARUR_{new}$
10: $Write_{task-set}\ [\,]\ in\ a\ file$
11: **else**
12: $convergence++$
13: **end if**
14: **end while**

(line 8 of algorithm 2). Then, its select the machine which has a maximum load on it. By taking the selected machine job mapping, load on the selected machine and execution time of each job on every machine, the load balancer returns the balancing task mapping of the selected machine *(lines 9-10 of algorithm 2).* The selected machine is removed from the total load *(lines 11-12 of algorithm 2).* After load balancing of all machine the while loop stops. The algorithm returns a balanced task mapping *(lines 13 of Algorithm 2).*

3.3 RALB-HC Load Balancer (Algorithm 3)

For the algorithm 3, the input is the execution time of all jobs on every machine, selected machine jobs, selected machine load and total load of the cluster. The output is the balance task mapping. The while loop starts with the condition if the total load on the selected machine is greater than the (total execution time/number of machines) the loop continue else its stops *(line 1 of algorithm 3).* From the selected machine jobs, select the minimum execution time job *(line 2 of algorithm 3).* Then it selects a new machine which has minimum execution time for the selected job other than the selected machine *(lines 3-4 of algorithm 3).*

Algorithm 2 : Resource Selector

Input: $Task\ mapping\ of\ sel-machine\ Jobs\ and\ Load[sel-machine]$
Output: Task-map[sel-machine]
1: **for** $i \leftarrow 0$ **to** number of machines **do**
2: $ET_{jobs[M_i]} \leftarrow ET_{J1} + + ET_{Jn}$
3: **end for**
4: $Total_{ET} \leftarrow \sum_{i=0}^{n} ET_{jobs[M_i]}$
5: **for** $i \leftarrow 0$ **to** number of machines **do**
6: $LoadM_i \leftarrow \frac{ET_{jobs[M_i]}}{Total_{ET}}$
7: $Total_{Load}[\,] \leftarrow LoadM_i$
8: **end for**
9: $t \leftarrow 0$
10: **while** $convergence < Con_{cric}$ **do**
11: $Select\ machine\ M_i\ with\ Maximum\ load\ from\ Total_{Load}$
12: $Task_{map}[\,] \leftarrow Load_{balancer}(ex_{time}[M_1, ..., M_n], Mi_{jobs}, Total_{load[Mi]})$
13: **Remove** $M_i\ Total_{load}[\,]$
14: t++
15: **end while**
16: **Return** $Task_{map}[\,]$

The algorithm 3 checks that the new selected machine has load less than (total execution time/number of machines) if true, then it mapped the select job on it and remove it from selected machine map jobs *(lines 5-7 of algorithm 3)*. If the new selected machine has a load more than the total execution time/number of machines, this machine is removed from the machine selection list *(lines 8-9 of algorithm 3)*. In this way, jobs are migrated to the new machine. The algorithm 3 returns a balanced mapping of the jobs for selected machine.

3.4 Evaluation Benchmarks

The RALB-HC is compared with the Resource aware load balancing algorithm (RALBA), Minimum Completion time (MCT), Min-Min, Max-Min, Resource-Aware Scheduling Algorithm (RASA), Sufferage and Task-Aware Scheduling Algorithm (TASA) [35]. The performance of the above-mentioned scheduling heuristic and proposed algorithm RALB-HC is evaluated using makespan, throughput and resource utilization metrics. Makespan is the latest finish time of any machine after executing the assigned task in a cluster environment. The better algorithm

Algorithm 3 : Load Balancer

Input: i. Execution time of all jobs on every machine.
ii. Selected Machine Jobs and Total Load on it.
iii. Total load on the cluster
Output: Task-map[sel-machine]

while $Total_{load[Mi]} > \frac{ET_{jobs[M_i]}}{Total_{ET}} * 100$ **do**
  $Selected_{job} \leftarrow$ **Select** $minimum\ time\ job\ from\ Mi_{job}$
  for $i \leftarrow 0$ **to** number of machines **do**
    $new_{machine} \leftarrow mig - machine$
    if $new_{machine} < \frac{Total_{ET}}{numberofmachines}$ **then**
      $new_{machine} \leftarrow maptask_{selected_job}$
      Remove $selected_{job} from Mi_{jobs}$
    else
      Remove $new_{machine}\ from\ selected_{job-all-machines-execution}$
    end if
  end for
end while
Return $Task_{map}[\,]$

has the lowest value of makespan. Average Resource Utilization Ratio (ARUR) is a measure of representing the resource utilization status of the cluster. The better algorithm has the value nearest to 1. Throughput is the number of jobs executed per unit time by the cluster. The better algorithm has the highest value of throughput [35].

Table 3.1: Evaluation of Benchmarks with Performance Metrics

S.No	Algorithm
1	RALBA
2	Minimum Completing Time (MCT)
3	Min-Min
4	Resource-Aware Scheduling Algorithm (RASA)
5	Sufferage
6	Task-Aware Scheduling Algorithm

$$makespan = max_{\forall j \in 1,2...m} (Mj_{ET}) \tag{3.1}$$

where m is the number of machines

$$throughput \ = \ \frac{n}{makespan} \tag{3.2}$$

where n is the number of the job.

$$ARUR \ = \ \frac{(\frac{\sum_{j=1}^{m} Mj_{ET}}{m})}{makespan} \tag{3.3}$$

To evaluate the classification (device suitability) and regression model (application estimator), we use standard metrics i.e. precision, recall, f measure and mean square error (MSE). The precision is the portion of correct positive classification (true positives) from cases that are predicted as positive [39]. The recall is the portion of correct positive classification (true positive) from cases that are actually positive[39]. F measure is the harmonic mean of precision and recalls [39].

$$Precision \ = \ \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.4}$$

$$Recall \ = \ \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.5}$$

MSE is used to evaluate the application estimator. MSE measures the square distance of forecast value and true value. MSE represents the model has high accuracy and high mean square error mean it has low accuracy [40].

$$F - measure = \ 2 \ * \ \frac{(Precision * Recall)}{(Precision + Recall)} \tag{3.6}$$

$$MeanSqaureError = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^*)^2 \tag{3.7}$$

Where n is the number of testing samples

$Y_i$is the actual value

Y_i^* is the predicted value

Chapter 4

Experimental Results and Discussion

4.1 Introduction

This chapter encompasses the details regarding implementation of RALB-HC with the detail of experimentation performed on three data sets, i.e. Scientific Application, synthetic and Google-like workload as shown in Figure 4.1. The three variations of the dataset are used to completely assess the proposed load balancer RALB-HC against state-of-the-art scheduling algorithms. These scheduling mechanisms include Task-Aware Scheduling Algorithm (TASA), Min-Min, Minimum Completion time (MCT), Max-Min and Resource-Aware Scheduling Algorithm (RASA). In the Scientific Application data set, the static analyzer is proposed that extract static features. Then, these features are used to train the device suitability predictor. The device suitability predictor predicts the best resource for the given job in the cluster. In addition to the device suitability predictor, application estimator is also trained by using static features and hardware features. The application estimator output is used by the RALB-HC for optimized load balancing. The synthetic workload is generated using a random-number comprising five categories of application (machine instruction (MI)) ranges, i.e. tiny

(1-250 MI), small (800-1200 MI), medium (1800-2500 MI), large (7000-10000 MI) and extra-large 30000-45000 MI). Moreover, we use the mechanism of Altaf et al [35] to generate a Google-like workload. Altaf et al. [35] have formulated the 5 categories for Google-like workload, i.e. small (15k-55k MI), medium (59k-99k MI), large (101k-135k MI), extra-Large 150k-337.5k MI) and huge (525k-900k MI). The proposed algorithm RALB-HC is evaluated using makespan, throughput and resource utilization metrics. Each experiment is explained in detail and evaluation is performed comprehensively.

4.2 Experimental Setup

In this research, three types of the data set are used, i.e. scientific application, synthetic and google-like workload. For scientific application data set, experiments are performed on a CPU-GPU system comprising on an Intel Core i5-4460 CPU and an NVIDIA GeForce GTX 760 GPU. All experiments are performed by using Linux Ubuntu 16.04 operating system. The specifications of the employed machine are presented in Table 4.1. Cloud simulator CloudSim is used for the synthetic and google-like workload. It is an open-source framework for the performance analysis. All the experiments are performed using 50 VM, hosted on 30 host machines within a data center.

TABLE 4.1: Experiment setup for the Scientific Application dataset

Device	**CPU**	**GPU**
Architecture	Haswell	Kepler
Base Clock	3.2 GHz	0.980 GHz
Boost Clock	3.4 GHz	1.033 GHz
Total Cores	4	1152 (CUDA cores)
Memory	8 GB	2 GB
Memory bandwidth	25.6 GB/s	192.2 GB/s
Performance (Single Precision)	409.6 GFLOPS	2257.9 GFLOPS
OpenCL SDK	Intel SDK for OpenCL 2016	CUDA 8.0
Compiler	GCC 5.4.0	Nvcc

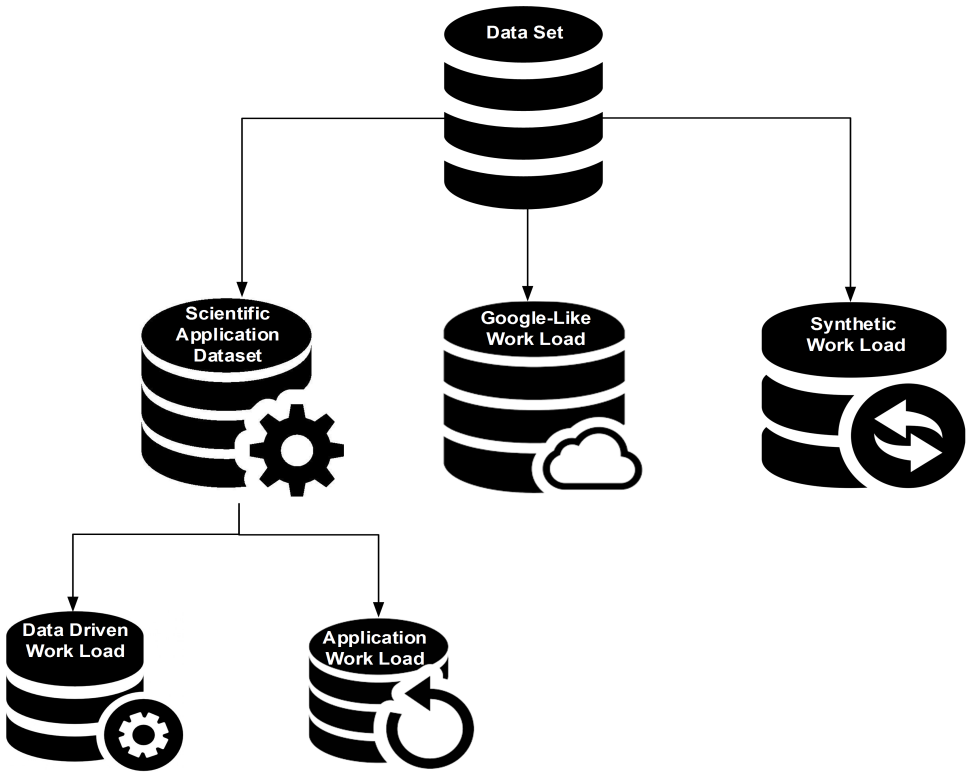

FIGURE 4.1: Explain the Comprehensive data set to test the effectiveness of the proposed algorithm RALB-HC. The scientific application data set is further split between Data Driven (data set having a variation of data size) Data sets and Application Driven (data set having number of applications). Google-Like workload and Synthetic workload is build according to recommendation of Altaf et el [35].

4.2.1 RALB-HC Implementation

We have chosen python language to implement the algorithm. Python is widely used in data science-related tasks. For a Scientific Application data set, the device suitability predictor and application estimator are also implemented in the python language. The sklearn library is used to implement the machine learning-based algorithms. The tree-based pipeline optimization library is used for device suitability and application estimator optimization task. These libraries are selected based on comprehensive analysis of all openly available libraries.

The work is made available on the GitHub[1] repository for further exploration. The source is written in python, because of its simplicity and its popularity among data

[1] https://github.com/usman189/RALB-HC

scientist and researchers in recent times. Our experience with python was very good and comfortable during this research study.

4.3 Scientific Application Data set

The Scientific Application dataset was collected by executing a total of 155 data parallel applications from mainstream benchmark suits (i.e., AMD, Polybench and self-employed) [3, 4, 13, 14, 22, 27, 66]. The benchmarks (shown in Table 6) are executed using multiple problem sizes resulting in a job pool of total 930 jobs. Each of 930 jobs is executed on two same and one different machine (1- GTX 780 and 2- GTX 740). Each machine has two processing units, i.e. CPU and GPU. The execution time of these three CPUs and three GPUs are profiled. A total of 930 profiled application is used to build a diverse training data for device suitability predictor and application estimator. The data parallel application code of these benchmarks is then provided to the code feature extractor, which extracts the features from data parallel application. The hardware specification is mentioned in Table 4.2 and Table 4.3.

TABLE 4.2: GTX 760 Machine Details

Device	**CPU**	**GPU**
Architecture	Haswell	Kepler (GTX 760)
Base Clock	3.2 GHz	0.980 GHz
Boost Clock	3.4 GHz	1.033 GHz
Total Cores	4 x 4 threads	1152 (CUDA cores)
Memory	8 GB	2 GB
Memory bandwidth	25.6 GB/s	192.2 GB/s
Performance (Single Precision)	409.6 GFLOPS	2257.9 GFLOPS
ISP	32	2
Memory Speed	1600mhz	6.0 Gbps

4.3.1 Experimental Dataset

Cummins et al. [34] have addressed that those predictive model that is trained on one benchmark suit fails to generalize across the other suite. By considering the

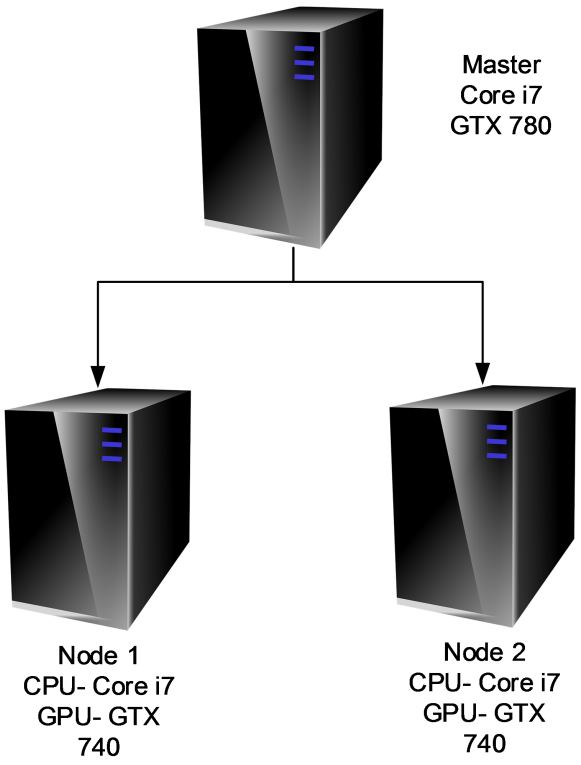

FIGURE 4.2: Scientific Application workload cluster composition

TABLE 4.3: GTX 740 Machine Details

Device	**CPU**	**GPU**
Architecture	Skylake i7-6700	GeForce (GT 740)
Base Clock	3.4 GHz	993 MHz
Boost Clock	4 GHz	1008 MHz
Total Cores	4 cores x 8 thread	384 (CUDA cores)
Memory	4 GB	2 GB
Memory bandwidth	34.1 GB/s	28.8 GB/s
Performance (Single Precision)	870.4 GFLOPS	762 GFLOPS
Instruction per clock	32	2
Memory Speed	2.133 GHz	1.8 Gbps

TABLE 4.4: Device suitability model tune parameters

Methods	**Hyper Tuning**
StackingEstimator	estimator=GaussianNB()
Features Selector	PCA (iterated-power=7 ,svd-solver="randomized")
Classification Model	XGBClassifier (learning-rate=0.5, max-depth=8 ,min-child-weight=1, n-estimators=100, nthread=1,subsample=0.5)

recommendations of Cummins et al [34], we have used two benchmark suites i.e. AMD and Polybench. The AMD and Polybench both benchmarks are extensively used in literature for heterogenous scheduling [3, 4, 13, 14, 22, 27, 66]. The detail

TABLE 4.5: Application time estimator tune parameters

Methods	**Hyper Tuning**
Normalizaiton	RobustScaler()
Regression Model	XGBRegressor(learning-rate=0.5, max-depth=6, min-child-weight=1, n-estimators=100, nthread=1, subsample=1.0)

of the benchmark used is mentioned in Table 4.6. These benchmarks are used to generate training data for our predictive models. These benchmarks cover the number of domains, including image processing, linear algebra and pattern recognition. In order to assess the effectiveness of the predictive model, leave one out cross-validation technique is used. The technique worked by selecting one benchmark for testing and then using the remaining for the training the model. Data details are mentioned in Table 4.11.

TABLE 4.6: Benchmarks

Suits	**Benchmark**	**Input Data Size**	**Versions**
AMD	Matrix Multiplication	1,769,472- 12,582,912	3
	Binomial Options	32,768- 294,912	9
	Bitonic Sortthe	32,768- 268,435,456	13
	Fast Walsh Transform	8,192- 221,184	17
	Matrix Transpose	131,072- 536,870,912	6
	Discrete Cosine Transformation	2,097,152- 1,887,436,800	17
	Floyd Warshall	524,288- 25,690,112	6
Polybench	3MM (3 Matrix Multiplications)	7,000,000 - 17,920,000	3
	GEMM (Matrix-multiply)	3,000,000- 27,000,000	5
	GESUMMV (Scalar, Vector and Matrix Multiplication)	8,012,000- 1,800,180,000	17
	MVT (Matrix Vector Product and Transpose)	4,016,000- 900,240,000	17
	ATAX (Matrix Transpose and Multiplication)	4,012,000- 900,180,000	17
	2MM (2 Matrix Multiplications)	5,000,000- 45,000,000	5
	2DCONV (2D Convolution kernel)	2000000- 1,568,000,000	17
	3DCONV (3D Convolution kernel)	1,000,000- 1,728,000,000	17
Own Developed	Matrix-Vector Multiplication	4,202,4961,514,299,392	16

4.3.2 Feature Extraction

The code feature extractor component extracts 30 distinct code features. The overview of the approach can be seen in Figure 3.1. The code feature sets represent the program behaviour. The purpose of the code feature extractor is to gather attributes of data parallel application code. Firstly, in order to ensure that the data parallel application code is error free, the data parallel application is just-in-time compiled using clang (front-end compiler) [65]. Then, the clang LLVM parser is used to extract features based on LLVM intermediate representation (IR) [65]. The python script (code feature extractor) uses Regular Expressions to detect features which are not available or can be detected by the LLVM (IR). In this research, we have only extracted code features and have not profiled the program. The full list of extracted features are mentioned in Table 4.7. After the feature extraction phase, both device suitability and application estimator model (trained offline) take in the feature vector and predicts device suitability as well as application estimator respectively. This prediction is provided with the proposed schedule for load balancing task.

Table 4.7: Data Parallel Application Code Features set

Index	Features Name	Index	Features Name
1	Data Size	16	Total number of Subtraction (Integer Data type)
2	Total number of Return statement	17	Total number of Function Call instruction
3	Total number of Control Statement	18	Total number of Functions
4	Total number of an Allocation instruction	19	Total number of Blocks
5	Total number of Load Instructions	20	Total number of Instructions
6	Total number of Store Instructions	21	Total number of Float Operation
7	Total number of Multiplication (Float Datatype) Operation	22	Total number of Integer Operation
8	Total number of Addition (Integer Datatype) Instruction	23	Total number of Loop Operation
9	Total number of Multiplication (Integer Datatype) Instruction	24	Base Clock (CPU/GPU)
10	Total number of Division (Float Datatype) instruction	25	Boost Clock (CPU/GPU)
11	Total number of Division (Integer Datatype) instruction	26	Total Cores (CPU/GPU)
12	Total number of Condition Check instruction	27	Memory (CPU/GPU)
13	Total number of Addition (Float Datatype) instruction	28	Memory bandwidth (CPU/GPU)
14	Total number of Addition (Integer Datatype) instruction	29	Performance (Single Precision) (CPU/GPU)
15	Total number of Subtraction (Float Datatype)	30	Instruction per clock (CPU/GPU)

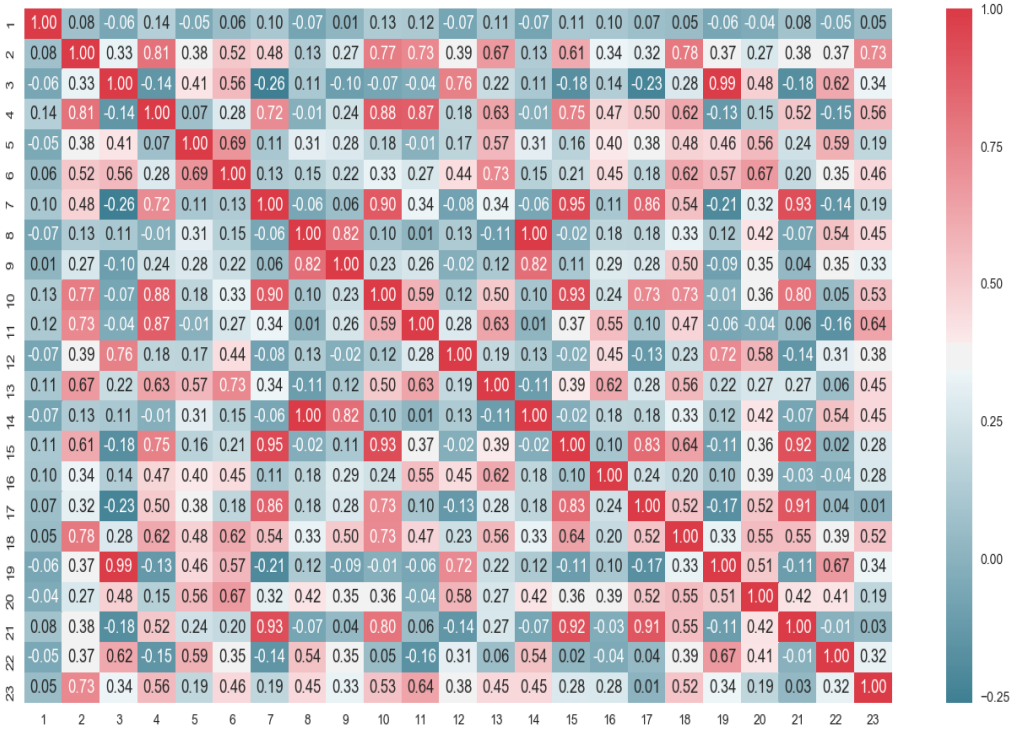

FIGURE 4.3: Correlation Analysis

4.3.3 Feature Selection

The feature set consists of 30 distinct features. Whether the dataset is collected by the non-domain-expert or it is provided by the domain experts, the selection of key attributes is very important. The motivation behind using less number of features (for predictive models) is the less redundant data, reduced overfitting issues, improved accuracy, and decreased training time of the algorithm. In this research, we have adopted two feature selection techniques, i.e., correlation analysis and tree-based feature selection. The correlation analysis provides an indication of the relative change between the two features. If a change in a features value causes the change in other features value too, then these features are referred to as co-related. The positively correlated feature is those features which change values in the same direction (i.e., increased or decreased). The features are denoted as negatively co-related if the change of one features value causes an inverse change in another features value. Figure 4.3 shows the correlation matrix of the employed code features (mentioned in Table 2).

The correlation analysis is very crucial for the selection of training features. If the data sample highly correlated features then, it will result in lower accuracy. The Figure 4.3 shows that the features [0] Data size, [6]- Total number of Multiplication (Float Datatype) Operation, [15]-Total number of Subtraction(Integer Datatype) instruction, [12]-Total number of Addition(Float Datatype) instruction, [16]-Total no of Function Call instruction, [8]- Total number of Multiplication (Integer Datatype) Instruction, [22]-Total number of Loop Operation and[20]-Total number of Float Operation has the smaller positive or a negative correlation with all other feature and they are ranked as top features by tree-based feature selection mentioned in Figure 4.4. The [10]-Total number of Division (Integer Datatype) instruction have very strong relationships with [1] - Total number of Return statement, [3]- Total number of Allocation instruction, [12]-Total number of Addition (Float Datatype) instruction and 14]-Total number of Subtraction (Float Datatype) instruction, which means that with the increase of [10]-Total number of Division (Integer Datatype) instruction, will tend to increase in the correlated feature set. After analyzing the correlation matrix and features importance ranking it was concluded that the features [5] - Total number of Store Instructions, [1] - Total number of Return statement, [17]-Total number of Functions, [3] - Total number of Allocation instruction and [10]-Total number of Division (Integer Datatype) instructions have very low or no contribution to output class so they are removed from feature set.

After correlation analysis and information gain analysis, selected features for the device suitability model are mentioned in Table 4.8 and for application execution time are mentioned in Table 4.9

4.3.4 Model Selection

In this research, two models have used, i.e. device suitability and application execution time predictor. The device suitability model predicts the application suitability on specific CPU_i or GPU_k. The application execution time predictor forecasts the execution time on all CPU_i and GPU_k. Both model type and

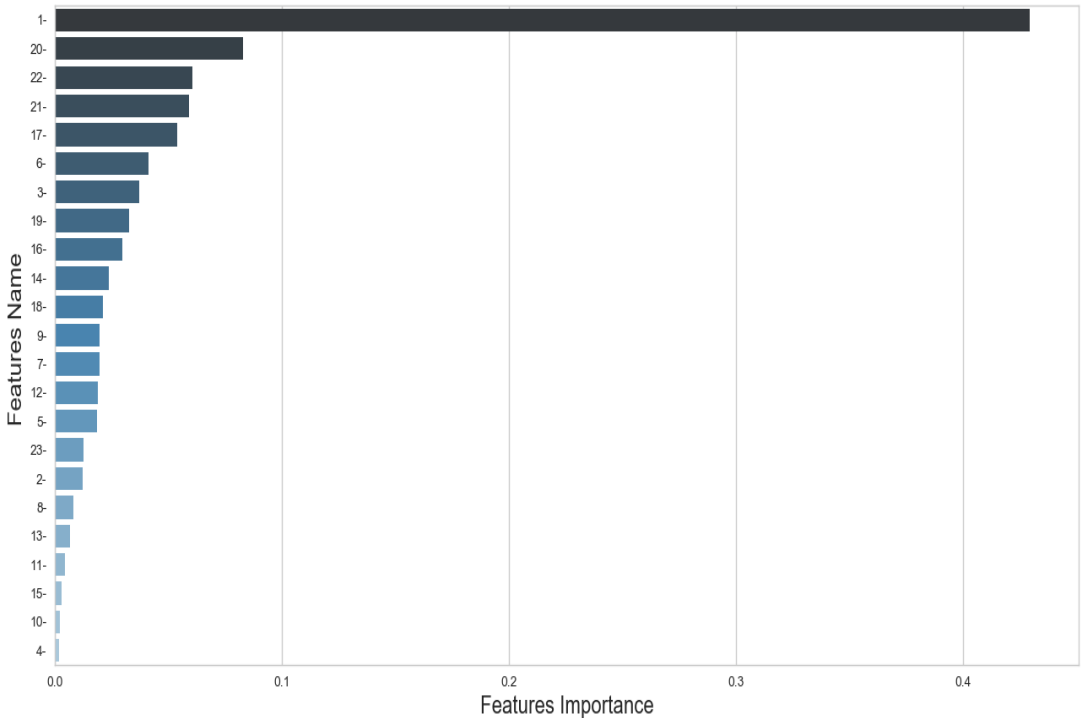

FIGURE 4.4: Tree Based Feature Selection

TABLE 4.8: Top Features set for predicting device suitability

Index	**Top Features Name**
0	Data Size
6	Total number of Multiplication (Float Datatype) Operation
8	Total number of Multiplication (Integer Datatype) Instruction
9	Total number of Division (Float Datatype) instruction
11	Total number of Condition Check instruction
12	Total number of Addition (Float Datatype) instruction
14	Total number of Subtraction (Float Datatype)
15	Total number of Subtraction (Integer Datatype)
16	Total number of Function Call instruction
18	Total number of Blocks
19	Total number of Instructions
20	Total number of Float Operation
21	Total number of Integer Operation
22	Total number of Loop Operation

its output are mentioned in Table 4.10. The device suitability model training methodology is shown in Figure 4.5. To train the device suitability classifier and application time estimator, it is required to label the training set of data parallel applications. Therefore, we label the output class for device suitability model by executing each application on all CPU_i and all GPU_k a device. The execution time of each data parallel application is noted for all available resources i.e. $CPU_i$and

TABLE 4.9: Top Features For Forecasting Application Execution Time

Index	**Top Features For Forecasting Application Execution Time**
23	Base Clock (CPU/GPU)
24	Boost Clock (CPU/GPU)
25	Total Cores (CPU/GPU)
26	Memory (CPU/GPU)
27	Memory bandwidth (CPU/GPU)
28	Performance (Single Precision) (CPU/GPU)
29	Instruction per clock (CPU/GPU)

GPU_k. The resource that consumes the lower execution time is labelled as a suited device for that data parallel application.

TABLE 4.10: Models Output Class

Model	**Type**	**Output**
Device Suitability Predictor	Classification	Model needs to predict application suitablilty on specific CPU_i or GPU_k
Application Execution time Estimator	Regression	Model needs to predict application execution time on all CPU_i and GPU_k .

For application estimator predictive model, a different approach was adopted. The execution time of each application on all CPU_i and all GPU_k a device is noted. The execution time of the devices is considered as output. The static code feature and device features are combined to make an input feature vector for the application time estimator.

TABLE 4.11: Training and Testing Data set

Set	**Instances**
Training Set	653 (70%)
Testing Set	277 (30%)
Total	930

In this research, we use the TPOT library [32] that chooses the right machine learning model and the best hyperparameter for that model. The TPOT is built on top of Scikit learn[32] that uses genetic programming to optimize our machine learning pipeline. After feature extraction and feature selection, TPOT does the feature

construction, model selection and hyperparameter tuning. Tree-based pipeline optimization is a new technique that shows significant promise for 1) making, machine learning tools more accessible to non-experts and 2) saving practitioners considerable amounts of time by automating the most tedious parts of machine learning. The labelled data (i.e., CPU or GPU) are provided to the TPOT classification class to determine the device suitability. Furthermore, the application estimator labelled data (execution time) is provided to the TPOT regression class to predict the potential application execution time on a device. Both the TPOT classes return hyper tuned model for both types of data, as shown in Table 4.4and Table 4.5.

The distinguishing feature of the proposed technique is that: on one hand it reduces the exponential growth of the time against conventional approaches. However, there is a general obsevation that the training time dratically influences the accuracy. For example, the increaseed amount of training time brings down the accuracy rate vise versa. Extreme gradient boosting algorithm is known for its reduced training time and high accuracy [44, 68]. Extreme gradient booting uses parallelization to use multicores of the processor. The use of parallelization results in reduced training time of the model. Extreme Gradient Boosting [44] is used for the classification and regression tasks, such as spam detection, face recognition, and financial predictions [42–45] etc. Gradient boosting technique is used by renowned search engines (i.e., Google, Bing, Yandex, and Yahoo etc.) Generally, grading boosting is used in search engines for web page ranking; however, this machine learning model is not limited to the application domain and can be adopted for various problems such as object detection and few more etc. [44, 45]. A tree base pipeline optimization technique performed hyper-parameter tuning which result in the selection of Extreme Gradient Boosting (Device Suitability) and Extreme Gradient Boosting Regression (application time estimator) algorithms.

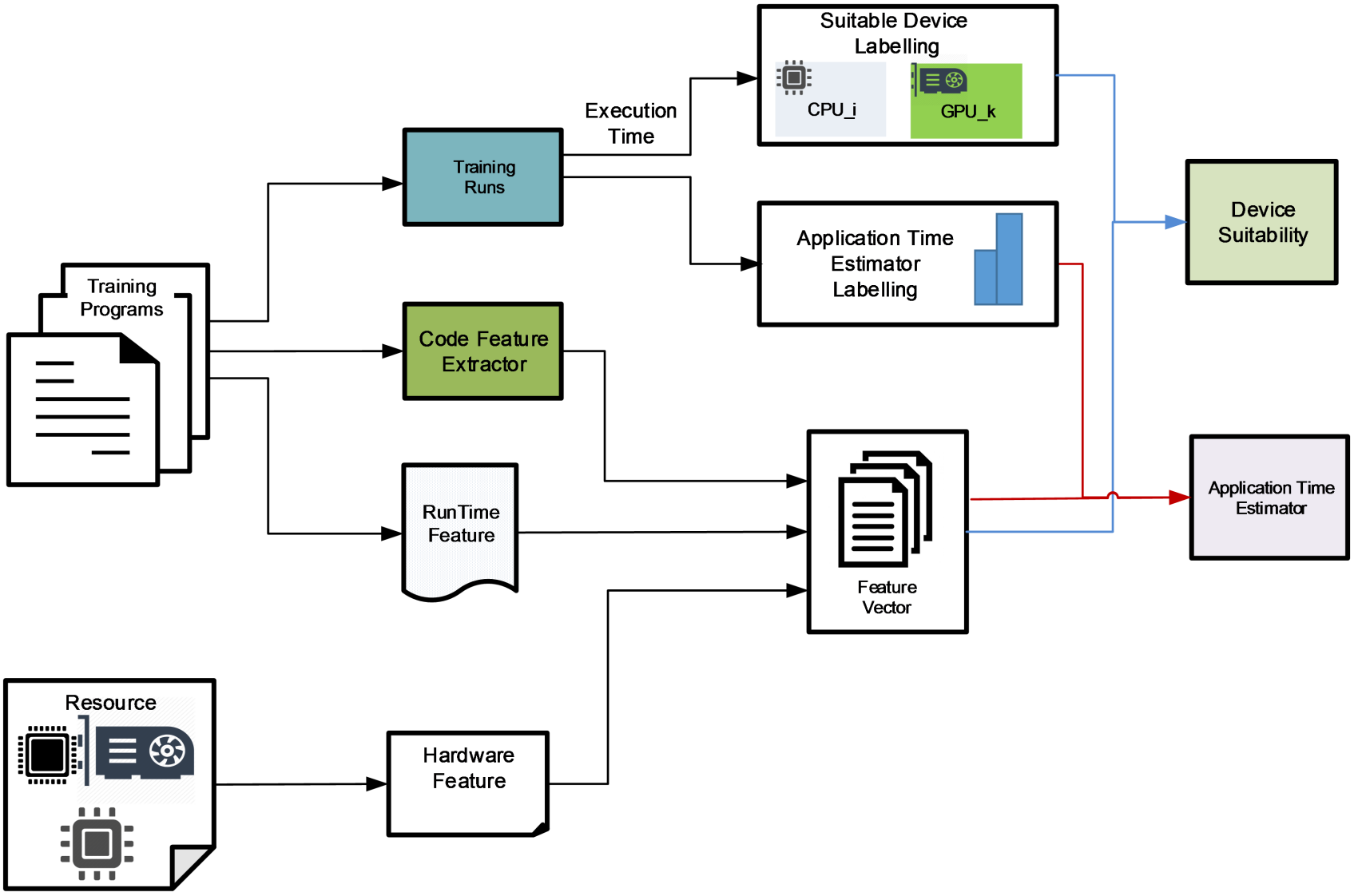

FIGURE 4.5: Device Suitability and Application Estimator Training

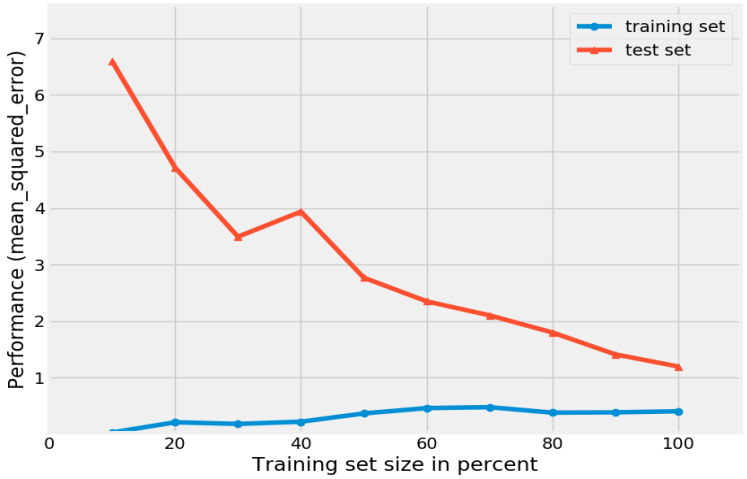

FIGURE 4.6: Application Execution Time Prediction Model

4.3.5 Model Training and Testing

After getting feature selection and model selection. The models are trained and tested on the dataset. The performance of both prediction models is mentioned in Figure 4.6 and Figure 4.7. The application estimator has a mean square error of 1.19 and device suitability model have an accuracy of 0.89.

Over the past few years, Receiver Operating Characteristic (ROC) curves [42–44]

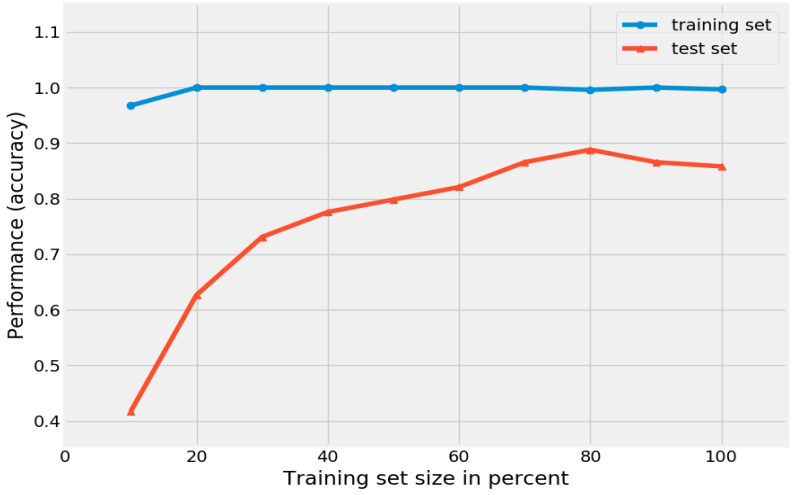

FIGURE 4.7: Device Suitability Prediction Model

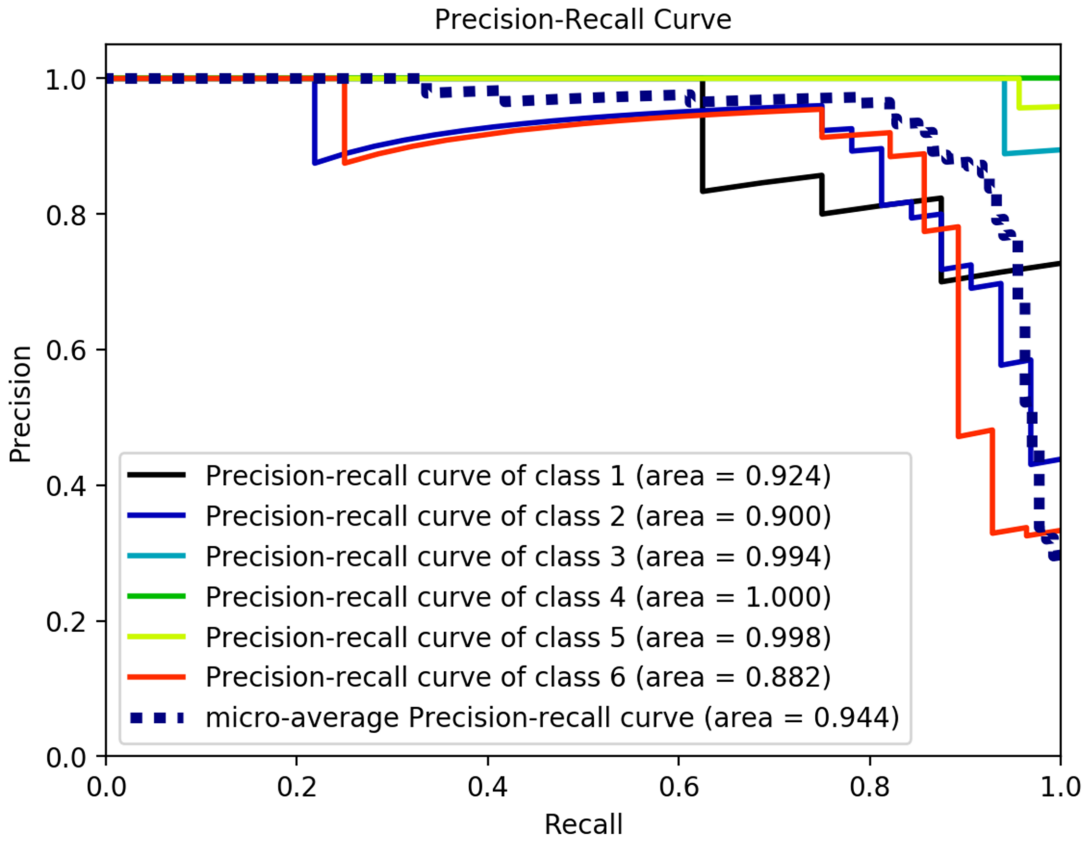

FIGURE 4.8: Precision-Recall Curve of Device suitability Model

are widely being used by many researchers to measure the performance of the trained machine learning models (Bradley, 1997). Furthermore, ROC curves are being used in various statistical methods that combine multiple clues, test results etc., and have been plotted and evaluated to represent a qualitative aspect of the trained machine learning models. ROC is basically a plot where True Positive Rate (TPR) is plotted on the Y-axis and False Positive Rate (FPR) is plotted on the X-axis. For every possible classification (i.e., output class), the TPR rate is based on the scenario where the actual classification is positive and the number of times the classifier has predicted positive results. The FPR determines that how often the classifier has incorrectly predicted positive when the actual classification is negative. Both the TPR and FPR range between 01 (0 means poor prediction whereas the value 1 means an accurate prediction). The area under the ROC (AUC), is widely utilized for weighing classier performance [42–44].

The results, based on the tree-based pipeline model is shown in Figure 4.8. The

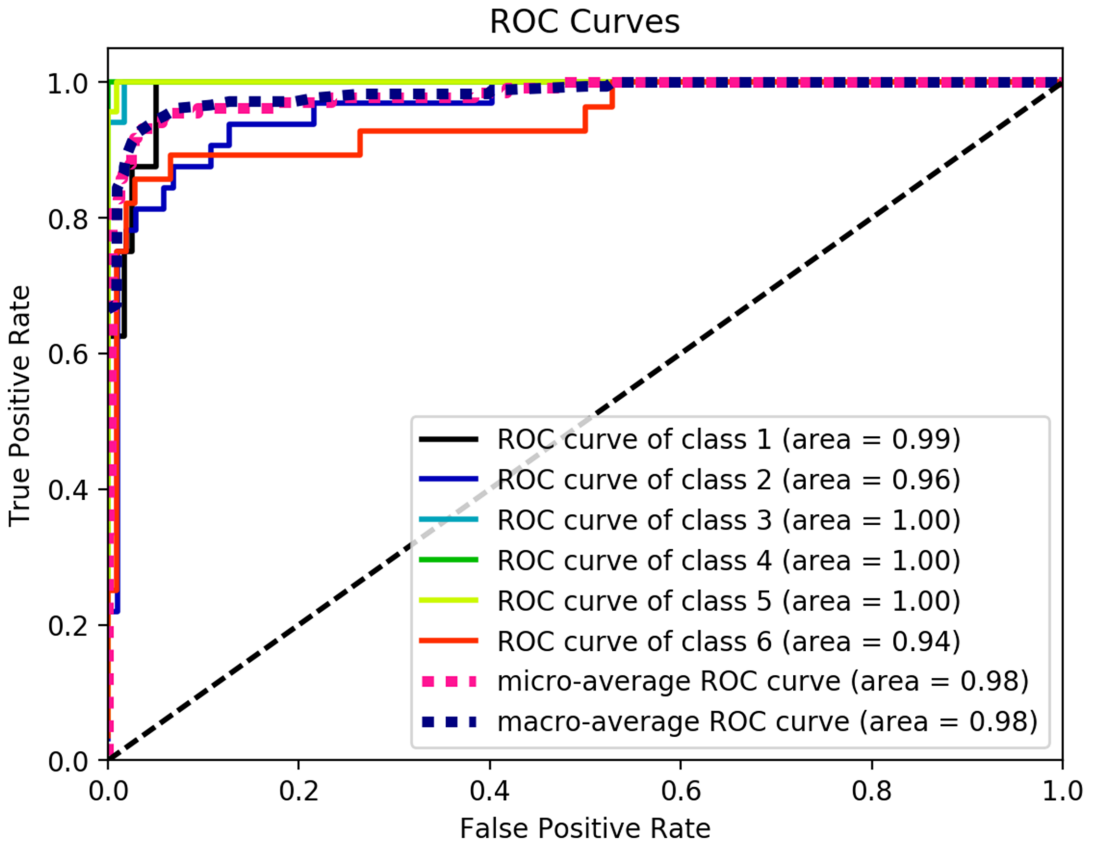

FIGURE 4.9: ROC Curve of Device Suitability Model

ROC curve for class 1 (GTX-740-1-CPU) is 0.92, class 2 (GTX-740-1-GPU) is 0.90, class 3 (GTX-740-2-CPU) is 0.99, class 4 (GTX-740-2-GPU) is 1 and class 5 (GTX-760-1-CPU) is 0.99. The high precision-recall curve value for all class signifies the excellent prediction. However, the precision-recall curve of class 6 (GTX-760-1-GPU) is 0.88.

The mean ROC curve for Device Suitability Classifier is shown in Figure 4.9. The mean ROC curve for the device suitability classifier is 0.98, however, the precision-recall curve of class 2 (GTX-740-1-GPU) is 0.96 whereas the class 3,4,5 is 1. The F-measure score of the device suitability model is 0.88. ROC depicts True Positive ratio (TPR) against the False Positive ratio (FPR) for different thresholds of the data using classification. The high ROC indicates the classifier produces good results. In Figure 4.10, the curve line is closer to the y-axis and the top border, this signifies that the detection rate of the device is significant as the TPR is high and the FPR is low. The high value of f-measure represents that the model

is capable of making fine distinctions. The high recall value represents that the correctly predicted classes are very high. The high precision for all classes indicates that the proportion of positive identifications is significantly high (min=0.81 and max =1)

The classification report of device suitability model on the Scientific Application dataset is shown in Figure 4.10. The F measure for class 1 (GTX-740-1-CPU) is 0.81, 0.84 for class 2 (GTX-740-1-GPU), 0.91 for class 3 (GTX-740-2-CPU) , 1 for class 4 (GTX-740-2-GPU) and 0.93 for class 5 (GTX-760-1-CPU). However, the F-measure score for class 6 (GTX-760-1-GPU) is 0.83. The average F-measure for the device suitability model is 0.88. The application estimator model means the square error is 1.19 and the R2 score is 0.81 as shown in Figure 4.11

4.3.6 Prediction Model Overhead

In propose model, the collection of benchmark suits took less than a day. Both prediction models are trained offline. The overhead of using device suitability predictor and application execution time estimator includes the feature extraction and making the predictions. The overhead of feature extraction is negligible (approximated 1s in total) as a feature is extracted at compile time. The prediction model training is performed once and it is a one-off-cost. The training time and testing time of both models are mentioned in Table 4.12. In total, the overhead of the prediction model is negligible i.e. 3 seconds. The computational complexity of heuristic is mentioned in Table 4.13. In experimentation and comparison, we didn't include the overhead cost of the proposed algorithm as well as the others.

TABLE 4.12: Training and Testing time

Model	**Training Time (seconds)**	**Testing Time (seconds)**
Device Sutability Model	1.09	0.007
Application Estimator Model	0.52	0.0048

TABLE 4.13: Computational Complexity : *M= Number of machine/processors, N= Number of jobs, n =If n is the number of jobs scheduled by Fill scheduler, then remaining N-n jobs will be scheduled by Spill scheduler.*

Heuristic	**Complexity**
MCT [69]	$O(MN)$
MinMin, MaxMin, RASA, TASA, and Sufferage [58, 59]	$O(MN^2)$
RALBA [35]	$O(M^2n + M.N - n)$
RALB-HC	$O(MN^2)$

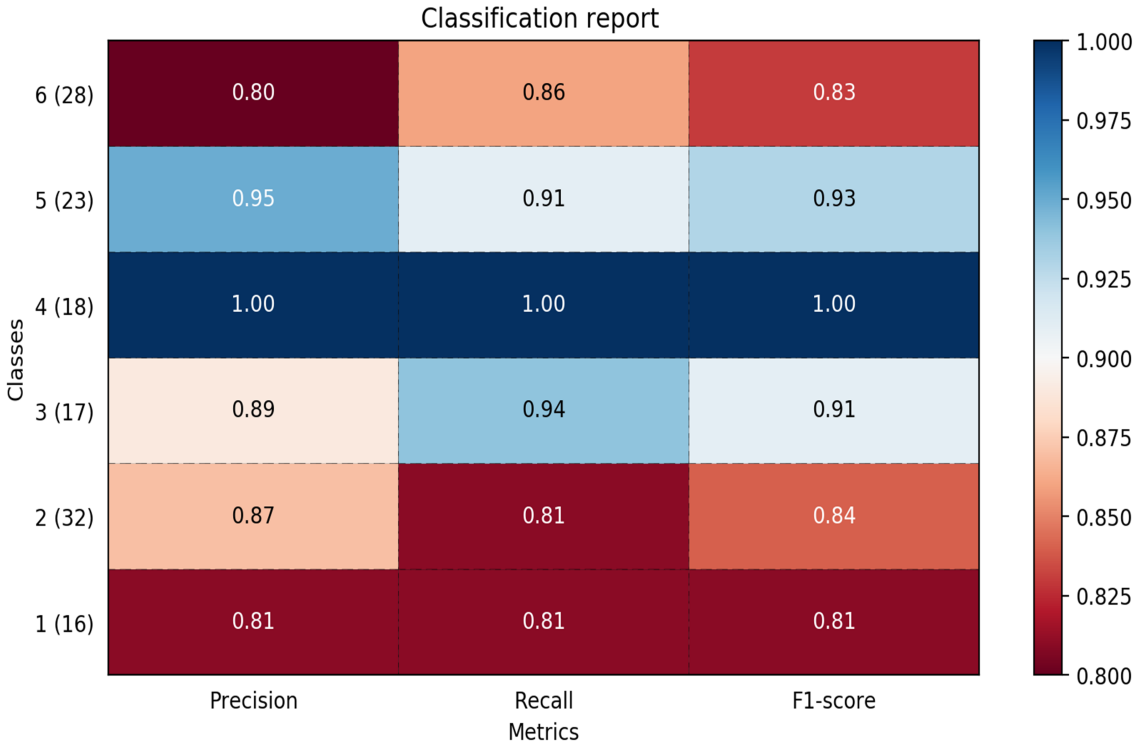

FIGURE 4.10: Classification Report of Device Suitability

4.3.7 RALB-HC Performance and Comparison

The device suitability and application model output is then used by the RALB-HC for load balanced scheduling. The same pool of application is given to all other scheduling algorithms. The test set for a Scientific Application data set is divided into two types. The data size driven and application driven. The data size driven is categorized into three different ranges, i.e. small (0-3000), medium (3000-13000) and large (13000-100000). The range in data size driven data represents N x N matrix size given by the user. The second type is application driven. It is also categorized into three types, i.e. small (0-200), medium (200-1200) and

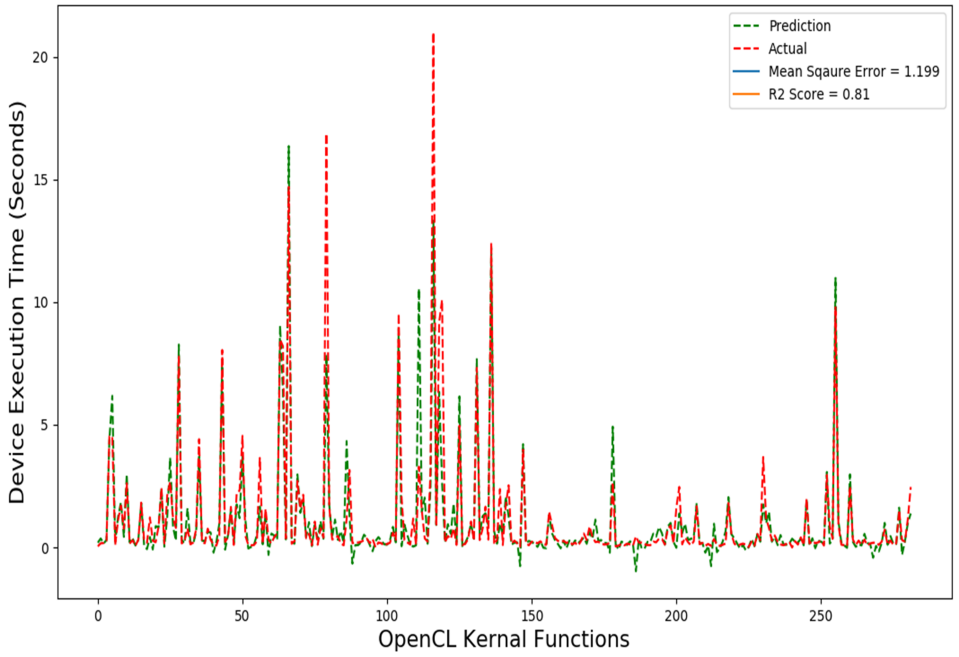

FIGURE 4.11: Application Estimator Model Mean Squared error

large (1200-6000). The range of application-driven data represent the number of application given to the scheduling.

Figure 4.12 shows the average execution time-based results for the data-driven job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 86.3 and 37.3 % reduced average execution time (for a job), respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC consumes on average 23.5% and 86.8% reduced execution time, respectively. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes on average 36.35% and 46.27%.

Findings: *When data-driven Scientific Application data set is used, RALB-HC consumes on average 52.79% reduced execution time.*

Figure 4.13 shows the average resource utilization ratio based results for the data-driven job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 99.83% and

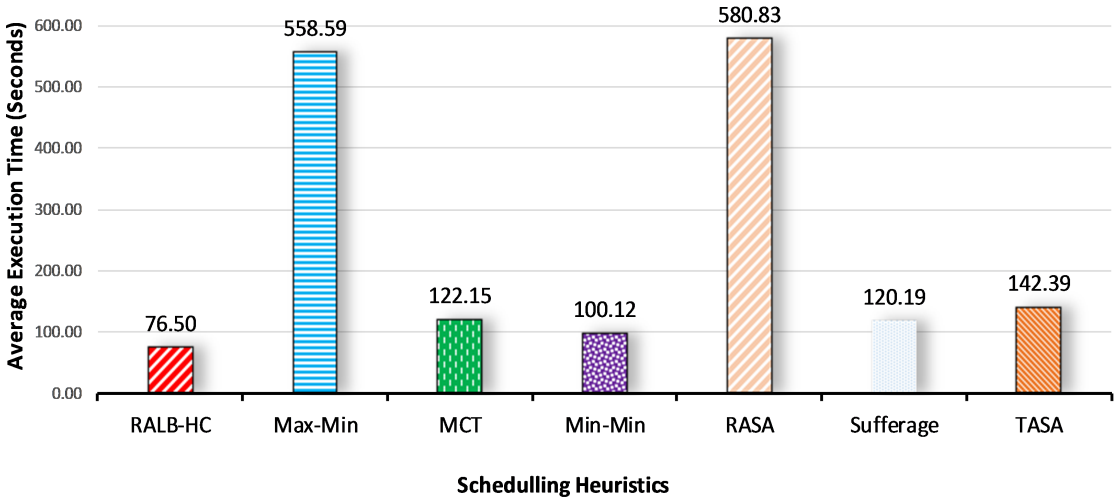

FIGURE 4.12: Data Driven Scientific Application Dataset: Average Execution Time

31.69% improved average resource utilization ratio, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC has on average 148.12% and 168.19% improved average resource utilization ratio. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes on average 30.79% and 128.93% respectively improved utilization of resources.

Findings: *When data-driven Scientific Application data set is used, RALB-HC results in 101.26% improved resource utilization ratio.*

Figure 4.13 shows the throughput (jobs per seconds) based on results for the data-driven job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 638.64% and 58.77% improved throughput respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC have on average 78.09% and 677.79% improved throughput. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes on average 57.28% and 90.59% respectively improved throughput.

Findings: *When data-driven Scientific Application data set is used, RALB-HC results in 266.86% improved throughput (jobs per seconds).*

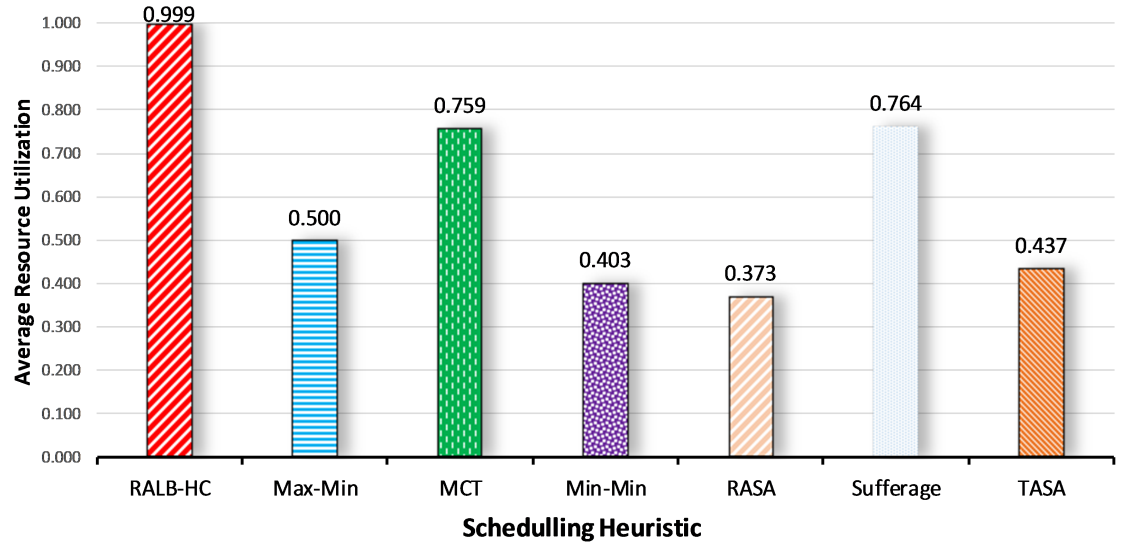

FIGURE 4.13: Data Driven Relistic Data set: Average Resource Utilization Ratio

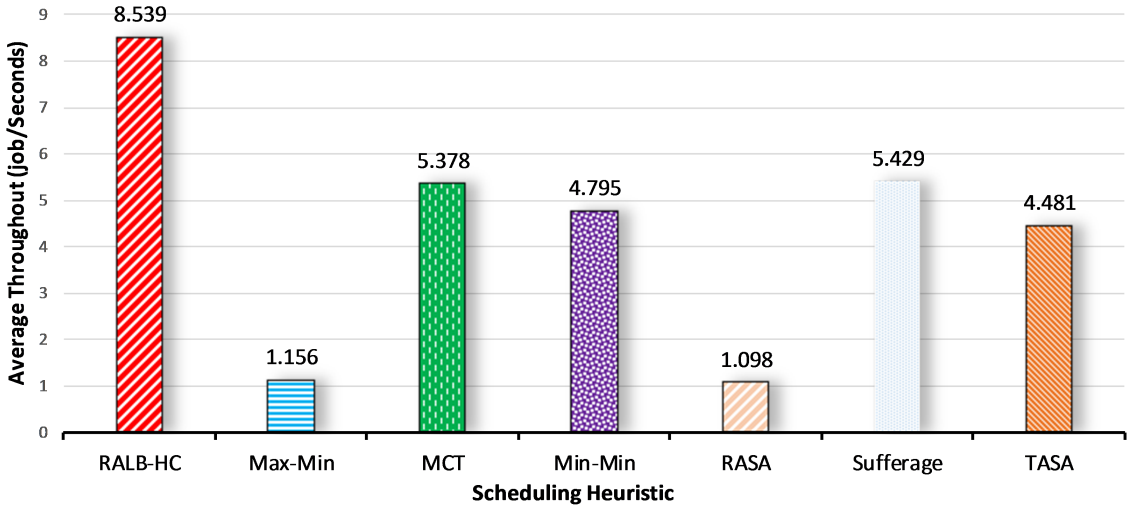

FIGURE 4.14: Data Driven Scientific Application Dataset: Throughput

Figure 4.15 shows the average execution time-based results for the application job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC , the proposed heuristic RALB-HC result in 24.08% and 32.84% reduced average execution time (for a job), respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC consumes on average 39.44 and 49.96% reduced execution time, respectively. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes

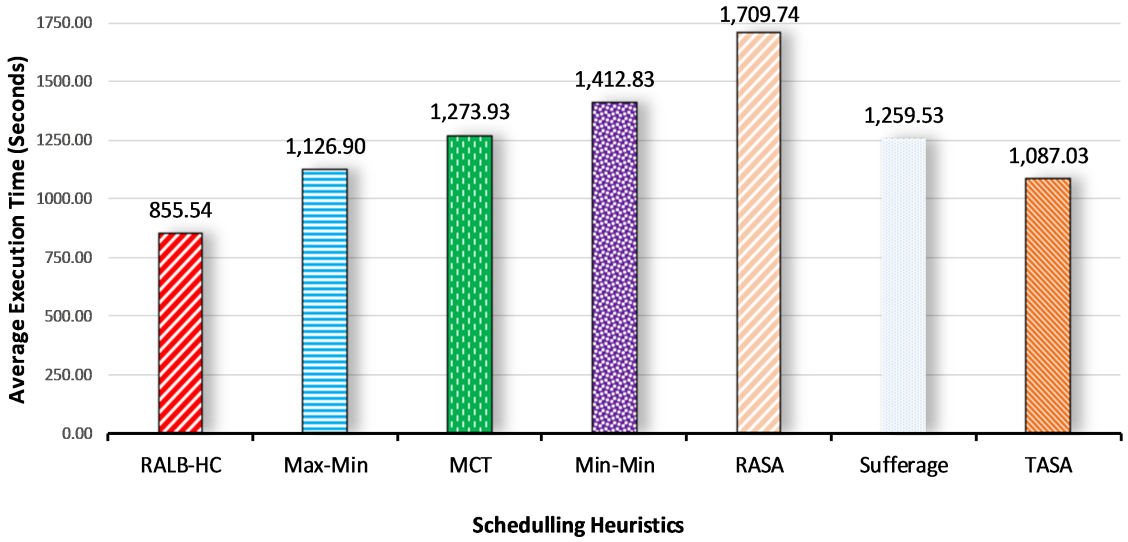

FIGURE 4.15: Applciation Driven Relistic Dataset: Average Execution Time

on average 32.07 and 21.30.

Findings: *When application driven Scientific Application data set is used, RALB-HC consumes on average 33.28 reduced execution time.*

Figure 4.16 shows the average resource utilization ratio based results for the application-driven job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 26.45% and 26.35 % improved average resource utilization ratio, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC has on average 92.30% and 63.19% improved average resource utilization ratio. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes on average 325.40% and 195.99% respectively improved utilization of resources.

Findings: *When data driven Scientific Application dataset is used, RALB-HC results in 71.61% improved resource utilization ratio.*

Figure 4.17 shows the throughput (jobs per seconds) based on results for the application-driven job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 60.26% and 47.6% improved throughput, respectively. When Min-Min and

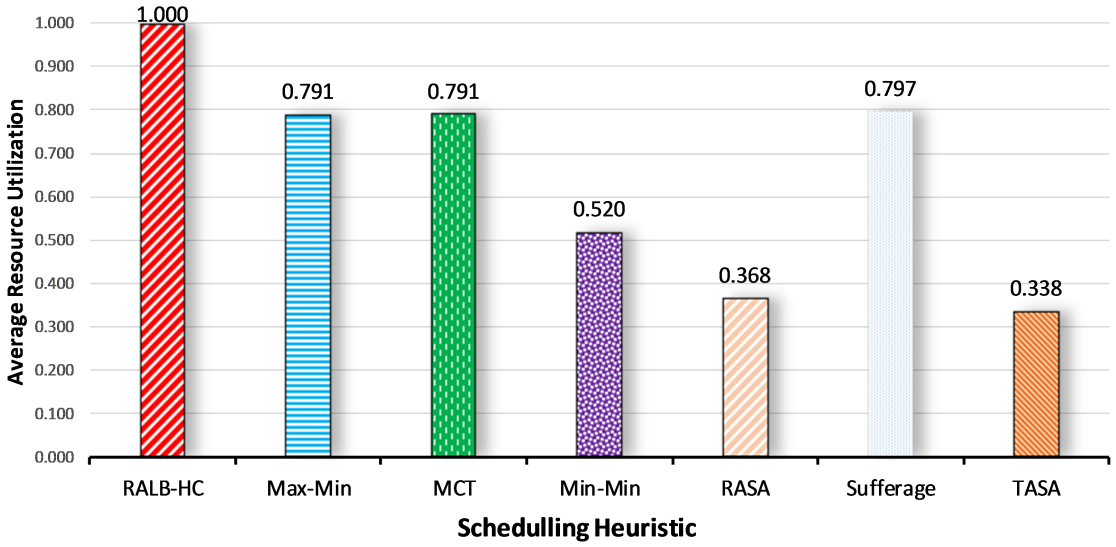

FIGURE 4.16: Application Driven Relistic Dataset: Average Resource Utilization Ratio

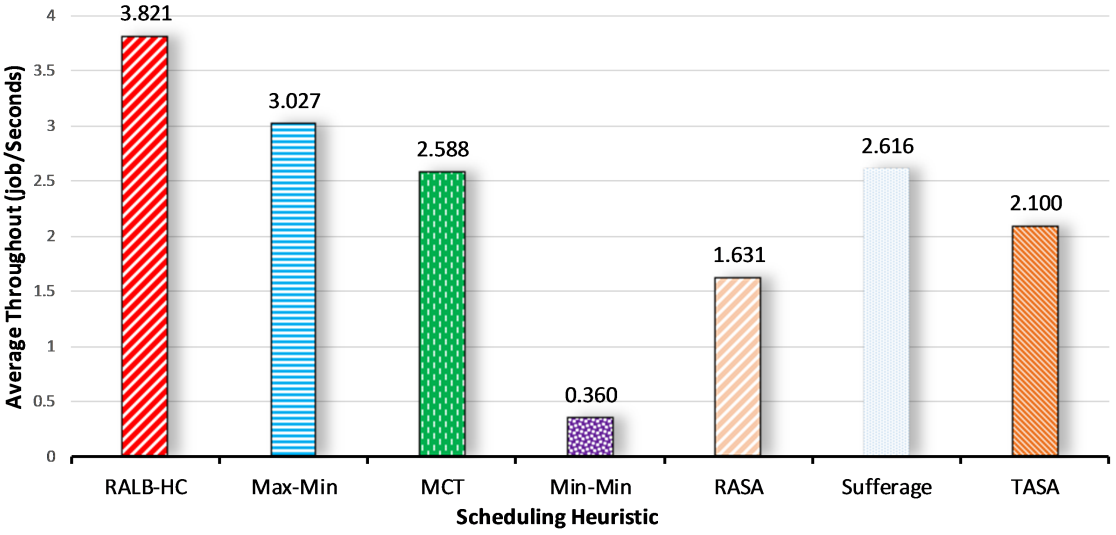

FIGURE 4.17: Application Driven Scientific Application Dataset: Throughput

RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC has on average 961.46% and 134.28% improved throughputs. As compared to the Sufferage and TASA, the proposed scheme RALB-HC consumes on average 46.06 and 81.97 respectively improved throughput.

Findings: *When data driven Scientific Application dataset is used, RALB-HC results in 266.86% improved throughput (jobs per seconds).*

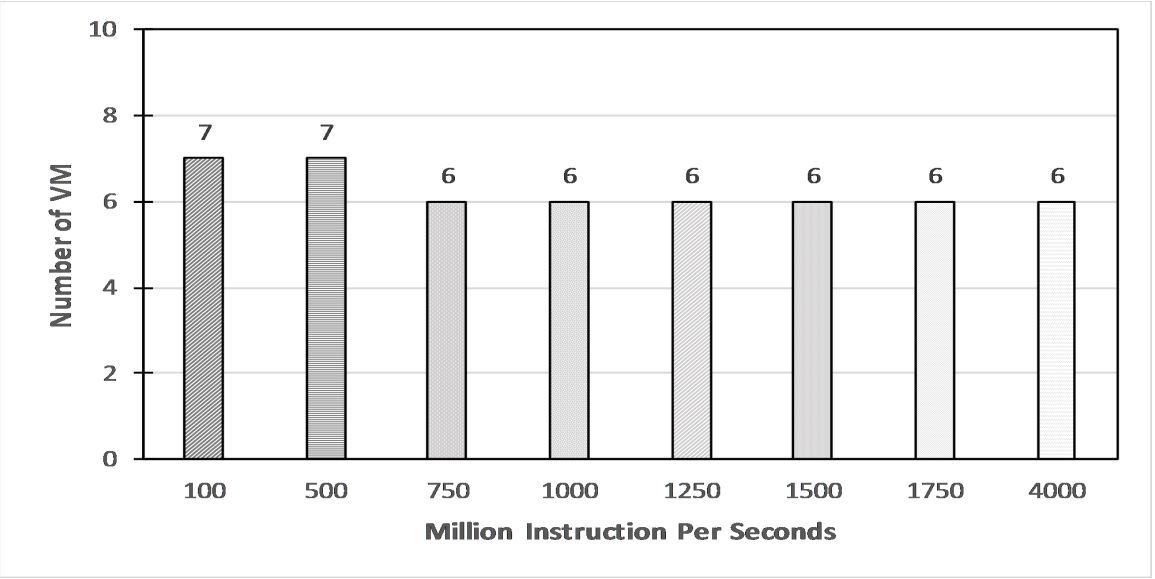

FIGURE 4.18: Virtual Machine Distribution

4.4 Google-like workload

We used the mechanism of Altaf et al for the generation of Google-like workload. Altaf et al. used the Monte-Carlo simulation method [35]. This scheme analyses the real-world traces and MapReduce logs from the M45 supercomputing cluster [35]. It is revealed that the majority of the application were of small size (execution time less than 15min) and a few were of large size (execution time over 300min) [35]. Based on these statistics, Altaf et al. have formulated the 5 categories for Google-like workload i.e. small (15k-55k MI), medium (59k-99k MI), large (101k-135k MI), extra-Large 150k-337.5k MI) and huge (525k-900k MI). The percentage distribution is shown in Figure 4.18. All the experiments are performed using 50 virtual machines (VM). The details of the VM machines are shown in Figure 4.18.

Figure 4.20 shows the average execution time-based results for the Google-like Work Load job pool. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 39.12% and 24.33% reduced average execution time (for a job), respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC consumes on average 29.90% and 27.90% reduced execution

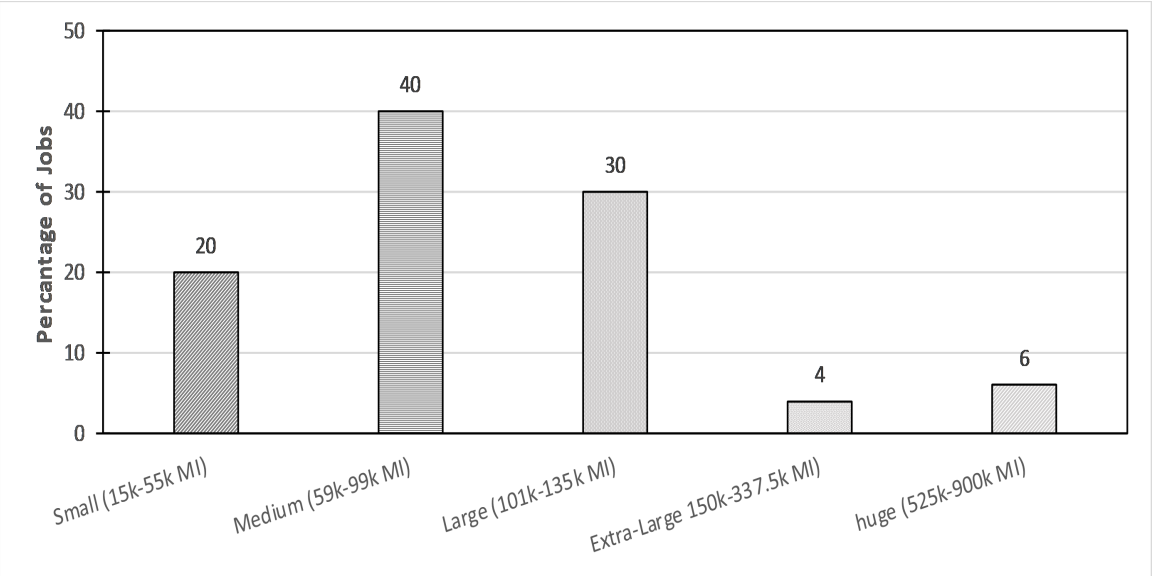

FIGURE 4.19: Google-like Work Load

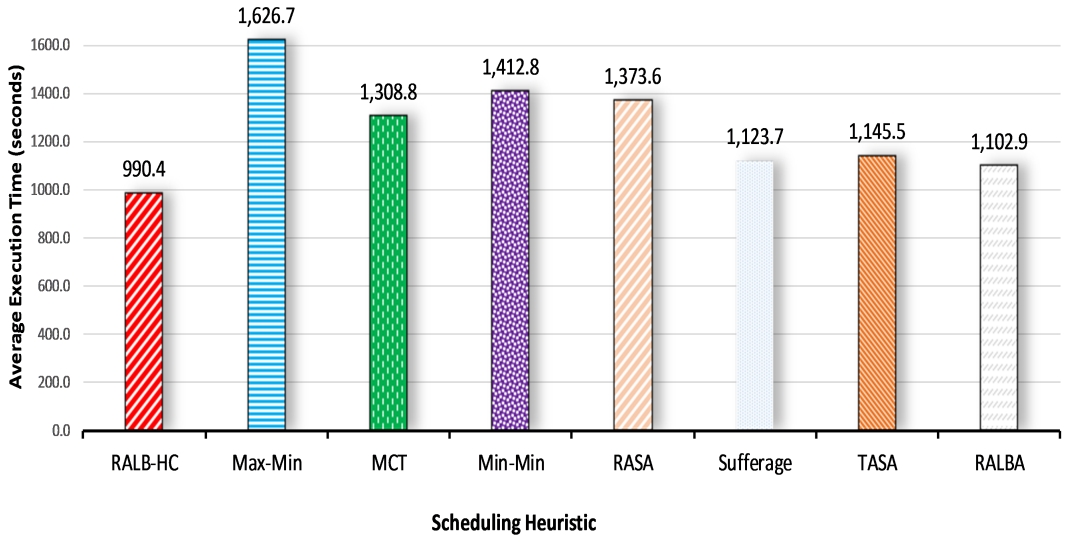

FIGURE 4.20: Google Like Work Load : Average Execution Time

time, respectively. As compared to the Sufferage, TASA and RALBA the proposed scheme RALB-HC consumes on average 32.07, 21.30 and 10.21.

Findings: *When the google-like Work Load is used, RALB-HC consumes on average, 22.41 reduced execution time.*

Figure 4.21 shows the average resource utilization ratio based results for the Google-like Work Load. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 14.07%

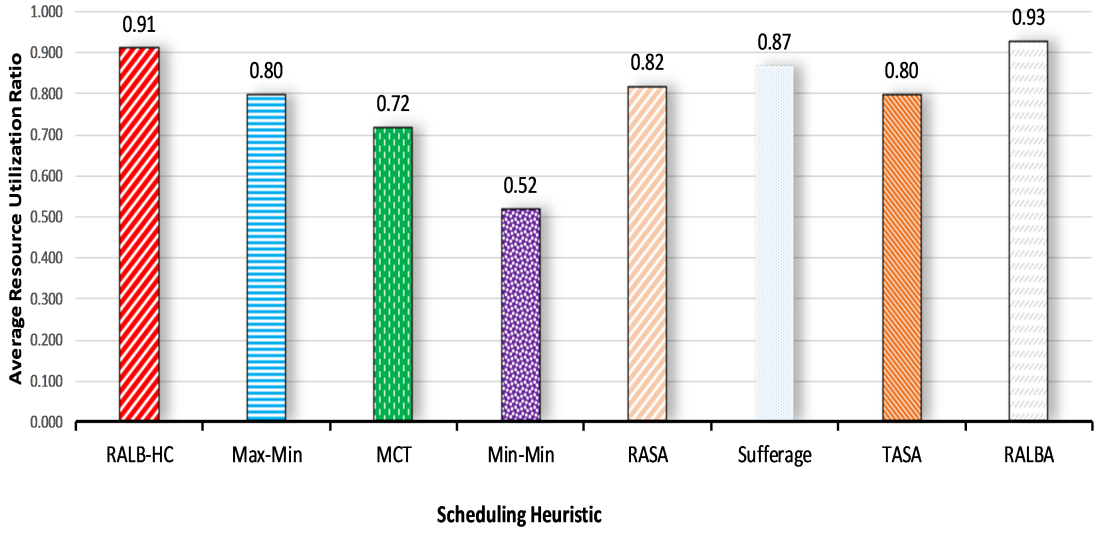

FIGURE 4.21: Google Like Work Load: Average Resource Utilization Ratio

and 26.74% improved average resource utilization ratio, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC has on average 75.49% and 11.29% improved average resource utilization ratio. As compared to the Sufferage and TASA, the proposed scheme RALB-HC achieved on average 4.89% and 14.07 % respectively improved utilization of resources. Whereas RALBA performed 1.88% better.

Findings: *When the google-like Work Load is used is used, RALB-HC results in 20.67% improved resource utilization ratio.*

Figure 4.22 shows the throughput (jobs per seconds) based on results for the Google-like Work Load. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 36.73% and 36.73 % improved throughput, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC have on average 51.93% and 21.54% improved throughput. As compared to the Sufferage, TASA and RALBA the proposed scheme RALB-HC achieved on average 16.37% , 18.90% and 11.62% improved throughput.

Findings: *When the google-like workload is used, RALB-HC results in 27.69 % improved throughput (jobs per seconds).*

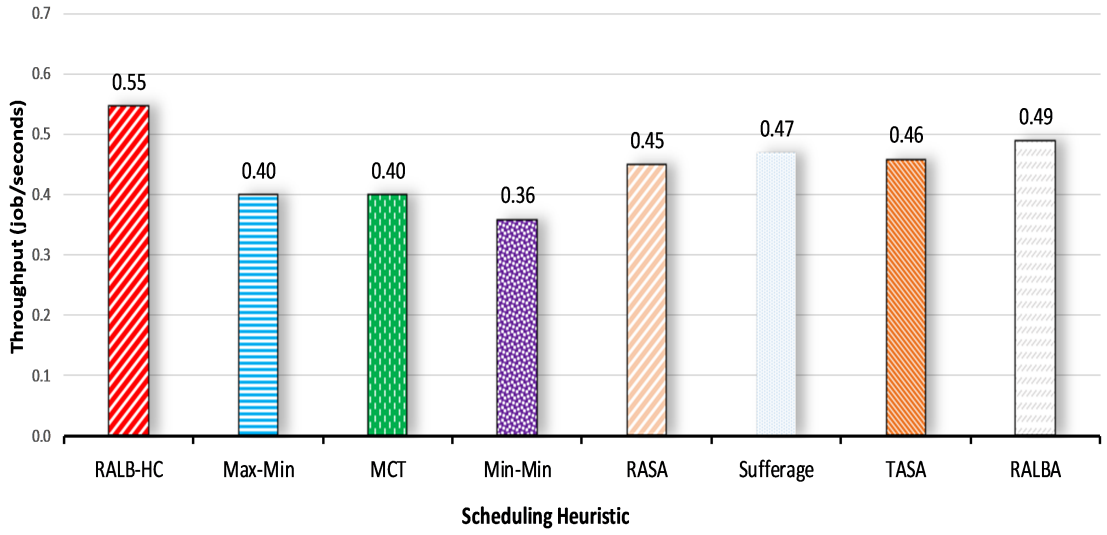

FIGURE 4.22: Google Like Work Load: Throughput

4.5 Synthetic workload

The synthetic data set is generated using random generation mechanism. All the experiments are performed using 50 virtual machines (VM). The details of the VM machines are shown in Figure 4.18. The synthetic workload is generated using a random-number comprising five categories of application (machine instruction (MI)) ranges, i.e. tiny (1-250 MI), small (800-1200 MI), medium (1800-2500 MI), large (7000-10000 MI) and extra-large 30000-45000 MI). The percentage distribution is shown in Figure 4.23.

Figure 4.24 shows the average execution time-based results for the Synthetic Work Load. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 65.81% and 17.15% reduced average execution time (for a job), respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC consumes on average 19.55% and 66.05% reduced execution time, respectively. As compared to the Sufferage, TASA and RALBA the proposed scheme RALB-HC consume on average 10.01%, 11.37% and 21.40% more.

Findings: *When Synthetic Work Load is used, RALB-HC consumes on average 17.97 reduced execution time. Whereas Sufferage, TASA and RALBA performed*

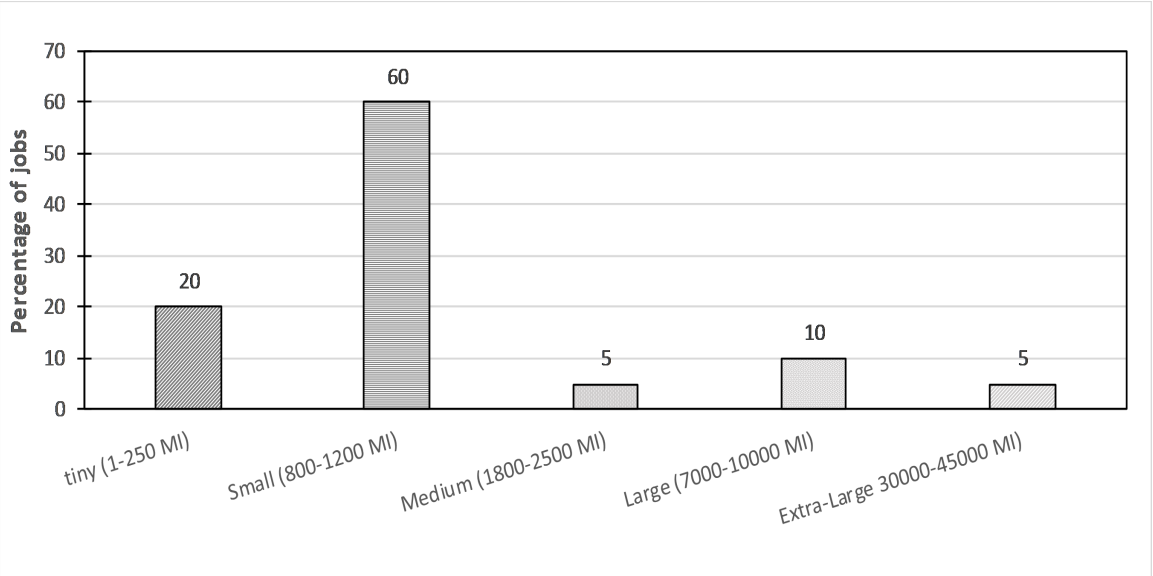

FIGURE 4.23: Synthetic Work Load

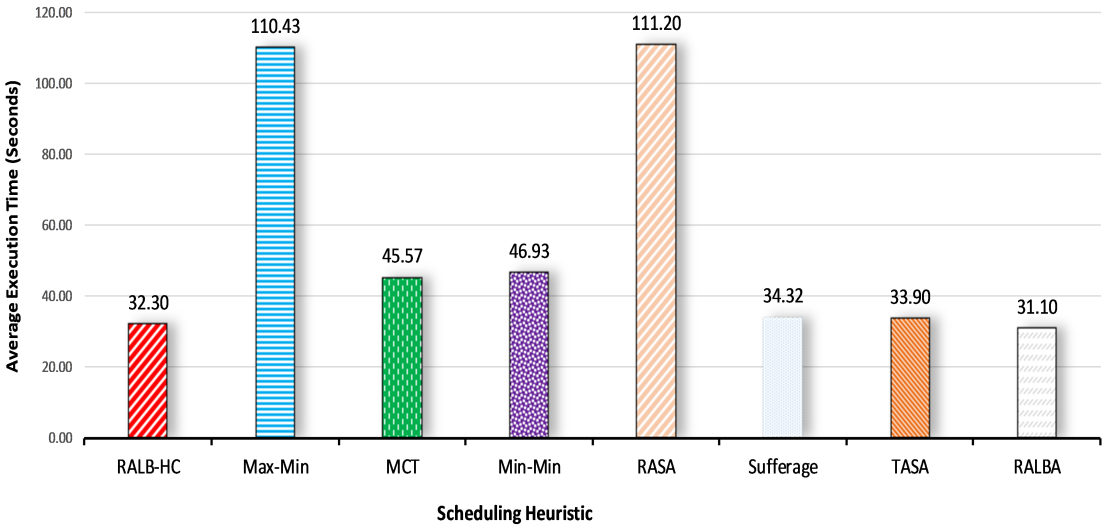

FIGURE 4.24: Synthetic Work Load : Average Execution Time

better.

Figure 4.25 shows the average resource utilization ratio based results for the Synthetic Work Load. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 123.01% and 82.87% improved average resource utilization ratio, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC has on average 147.12% and 140.61% improved average resource

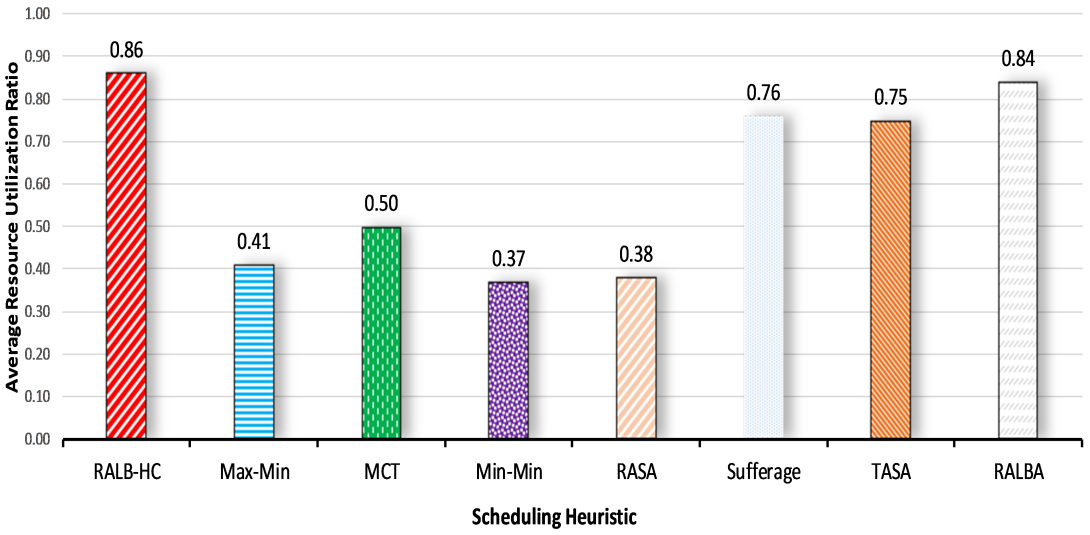

FIGURE 4.25: Synthetic Work Load: Average Resource Utilization Ratio

utilization ratio. As compared to the Sufferage, TASA and RALBA the proposed scheme RALB-HC achieved on average 20.31

Findings: *When Synthetic Work Load is used, RALB-HC results in 77.81% improved resource utilization ratio.*

Figure 4.26 shows the throughput (jobs per seconds) based on the results of the Synthetic Work Load. When the Max-Min and MCT scheduling mechanism is compared with the RALB-HC, the proposed heuristic RALB-HC result in 188.84% and 63.16% improved throughput, respectively. When Min-Min and RSA scheduling mechanism is compared with the RALB-HC, the proposed scheme RALB-HC have on average 69.13% and 199.13% improved throughput. As compared to the Sufferage, TASA and RALBA the proposed scheme RALB-HC achieved on average 23.49% , 21.94% and 14.80% improved throughput.

Findings: *When Synthetic Work Load is used, RALB-HC results in 82.93% improved throughput (jobs per seconds).*

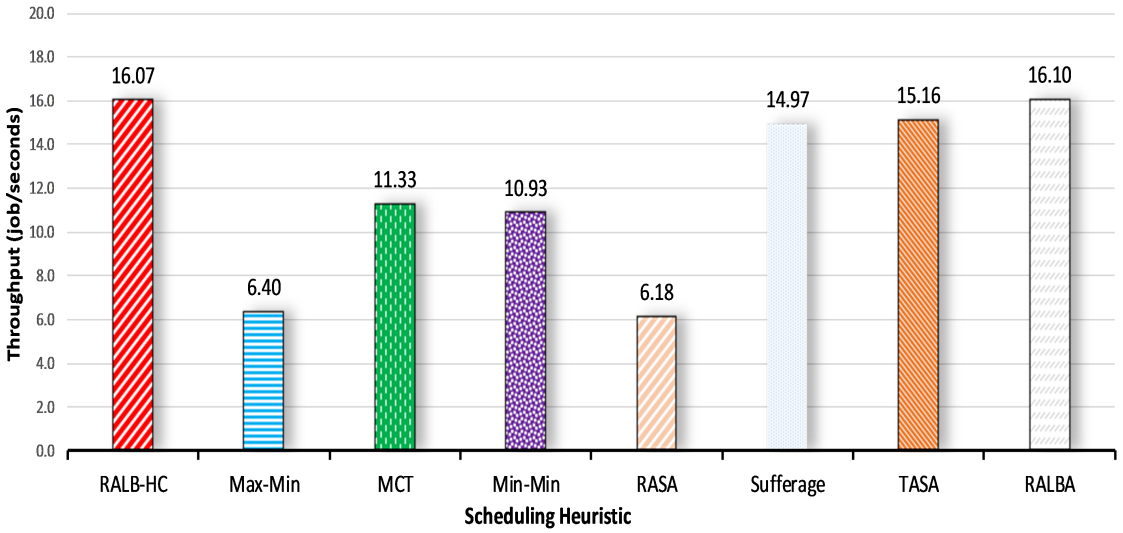

FIGURE 4.26: Synthetic Work Load: Throughput

4.6 Performance Discussion

Experimental results (presented in Figures 4.12 to Figure 4.25) show that the proposed scheduling mechanism RALB-HC outperforms all the other scheduling schemes with respect to execution time (for complete job pool), throughput, and average resource utilization ratio performance metrics. The improved performance of the proposed RALB-HC scheduling heuristic is due to the indulgence of machine learning model. First of all, all the jobs (in the job pool) are categorized according to device suitability. Next, in order to improve load balance, jobs are mapped to the migrated machine. The load balanced and device suited application mapping mechanism of the proposed RALB-HC scheduling heuristic results in reduced execution time (for the job pool), higher system throughput, and higher device utilization as compared to the other scheduling heuristics. The state-of-the-art scheduling schemes Minimum Completion time (MCT), Min-Min, Max-Min, Resource-Aware Scheduling Algorithm (RASA) and Task-Aware Scheduling Algorithm (TASA) are unaware of the device suitability of the applications resulting in a sub-optimal performance as compared to the RALB-HC.

Furthermore, the research questions were the following:

1. RQ-1: How to design and develop a load balancing scheduling algorithm to achieve the minimal execution time, maximal throughput and improved resource utilization?

2. RQ-2: To analyse optimization technique for design device suitability classifier and execution time predictor?

    2.1. RQ-2.1: Which set of features plays an important role to predict data parallel application device suitability?

    2.2. RQ-2.2: Which are the most important factors, forecasting the execution time of data parallel application?

The answer to the first question is that the proposed algorithm RALB-HC performed better. Load balanced is induced between heterogeneous devices by incorporating application time execution estimation, on a potentially suitable device, in the scheduling decisions. Experimental results on a large set of applications show that the RALB-HC reduce execution time by 31.61% in comparison to the baseline scheduling scheme. Moreover, RALB-HC achieved 67.8% and 147.35% improved resource utilization ratio and throughput.

The answer to the 2nd question is that Extreme gradient boosting classifier and regression model is better to predict device suitability and application estimation model. We used the tree base pipeline model tune parameter of the classification and regression model. The Extreme boosting algorithm results in the best performing model. It is fast, memory efficient and of high accuracy.

We used tree-based feature selection and correlation analysis to answer 2nd question (2.1). The data size of the job is the most impacted features while operation (multiplication, division, number of blocks and number of instruction also plays a significant role to classify job to the most suited device. It is revealed that Table 4.8 features play an important role. For part (c), it is concluded that the hardware features play a significant role in bringing the mean square error to 1.91 and R2 score to 0.81. The hardware features are mentioned 4.9

The answer to the fourth question

Chapter 5

Conclusion and Future Work

In heterogeneous cluster environment, programmers map application to specific devices. This decision is not optimal in a multi-node or cluster of a heterogeneous system. The number of jobs is submitted to the scheduler. The scheduler maps the application to the computing devices. The decision about the work distribution should be balanced to achieve maximal throughput. It is very difficult for a programmer to decide the mapping of jobs to a variety of heterogeneous computing devices.

In this research, a novel job scheduling mechanism, named as RALB-HC that uses machine learning to classify applications according to their device suitability. Moreover, RALB-HC also predicts the achievable execution time of each submitted job when executed on the preferred device. After that RALB-HC schedule batch of jobs in a load balanced manner while effectively utilizing heterogeneity that is inherent to heterogeneous computing devices. In particular, the following are the main contributions of the research:

1. In-depth analysis of the state-of-the-art scheduling mechanisms for heterogeneous machines to identify the merits and demerits of several existing heuristics;

2. A novel machine-learning based scheduling heuristic that considers device suitability and application time estimation of computing devices for scheduling and execution of a job pool in a load balanced manner on heterogeneous machines;

3. A development of a machine learning based classifier to decide a jobs suitability for a particular heterogeneous cluster.

4. Machine learning based application execution time prediction of a job due to execution on a suitable device.

5. Empirical investigation and comparison of the proposed scheduling technique with state-of-the-art scheduling heuristics from literature, demonstrating significant improvement in execution time, throughput, and resource utilization.

5.1 Future Work

In this research, a novel Resource-Aware scheduling for Heterogeneous Cluster (RALB-HC) is proposed that distributes workload based on resource computing capability and type of application. RALB-HC determine which data parallel application are like to best utilize a device. The application execution time has also been forecast. The device prioritized list and resource performance estimation are then used by a proposed *Resource aware load balancer for the heterogeneous cluster (RALB-HC)*. The prediction model, as well as the convergence factor, helps RALB-HC to achieve optimized results. The significant contribution of proposed RALB-HC is that it consumes same amount of comptutational cost when compared with traditional heuristic appraoches. However, a exception has been recorded when results were compared with RALBA [35]. The experimental results show that RALB-HC on average achieved 31.61% reduced execution time, 67.8%

improved average resource utilization ratio and 147.35% improved throughput. In future extension the RALB-HC framework can further be extended for incorporating Green Energy as an optimization parameter and can be utilized in sensor networks.

Bibliography

[1] G. S. John E. Stone, David Gohara, OpenCL: A ParalleL Programming Standard for HeterogeneouS Computing Systems, Comput. Sci. Eng., vol. 12, no. 3, pp. 6673, 2010.

[2] M. K. Emani, Z. Wang, and M. F. P. OBoyle, Smart, adaptive mapping of parallelism in the presence of external workload, Proc. 2013 IEEE/ACM Int. Symp. Code Gener. Optim. CGO 2013, 2013.

[3] A. Ghose, S. Dey, P. Mitra, and M. Chaudhuri, Divergence Aware Automated Partitioning of OpenCL Workloads, Proc. 9th India Softw. Eng. Conf., pp. 131135, 2016.

[4] D. Grewe and M. F. P. OBoyle, A static task partitioning approach for heterogeneous systems using OpenCL, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 6601 LNCS, pp. 286305, 2011.

[5] C.-K. Luk, S. Hong, and H. Kim, Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping, Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture - Micro-42, p. 45, 2009.

[6] D. Grewe, Z. Wang, and M. F. P. M. OBoyle, Portable mapping of data parallel programs to OpenCL for heterogeneous systems, Code Gener. , vol. V, no. 212, pp. 110, 2013.

[7] M. E. Belviranli, L. N. Bhuyan, and R. Gupta, A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures, ACM Trans. Archit. Code Optim., vol. 9, no. 4, pp. 120, 2013.

[8] H. J. Choi et al., An efficient scheduling scheme using estimated execution time for heterogeneous computing systems, J Supercomput, vol. 65, pp. 886902, 2013.

[9] C. Gregg, J. Brantley, and K. Hazelwood, Contention-Aware Scheduling of Parallel Code for Heterogeneous Systems, USENIX Work. Hot Top. Parallelism, p. None, 2010.

[10] V. T. Ravi and G. Agrawal, A dynamic scheduling framework for emerging heterogeneous systems, 2011 18th Int. Conf. High Perform. Comput., pp. 110, 2011.

[11] Augonnet, C., Thibault, S., Namyst, R. and Wacrenier, P.A., 2011. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience, 23(2), pp.187-198.

[12] A. Ghose, L. Dokara, S. Dey, and P. Mitra, A Framework for OpenCL Task Scheduling on Heterogeneous Multicores, Parallel Process. Lett., vol. 27, no. 03n04, p. 1750008, 2017.

[13] K. Kofler, I. Grasso, B. Cosenza, and T. Fahringer, An Automatic Input-Sensitive Approach for Heterogeneous Task Partitioning Categories and Subject Descriptors, Proc. 27th Int. ACM Conf. Int. Conf. Supercomput. - ICS 13, pp. 149160, 2013.

[14] Y. Wen and M. F. P. OBoyle, Merge or Separate, Proc. Gen. Purp. GPUs - GPGPU-10, pp. 2231, 2017.

[15] A. M. Aji, A. J. Pea, P. Balaji, and W. chun Feng, MultiCL: Enabling automatic scheduling for task-parallel workloads in OpenCL, Parallel Comput., vol. 58, pp. 3755, 2016.

[16] J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, and J. Lee, SnuCL: an OpenCL framework for heterogeneous CPU/GPU clusters, Proc. 26th ACM Int. Conf. Supercomput. - ICS 12, p. 341, 2012.

[17] R. Pandit, Prasanna and Govindarajan, Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices, Proc. Annu. IEEE/ACM Int. Symp. Code Gener. Optim., p. 273, 2014.

[18] M. Becchi, S. Byna, S. Cadambi, and S. Chakradhar, Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory, Proc. 22nd ACM Symp. Parallelism algorithms Archit., pp. 8291, 2010.

[19] P. Huchant, M. C. Counilh, and D. Barthou, Automatic OpenCL task adaptation for heterogeneous architectures, Eur. Conf. Parallel Process., vol. 9833 LNCS, pp. 684696, 2016.

[20] Ra. Perez, Borja; Bosque, Luis; Beivide, Simplifying Programming and Load Balancing of Data Parallel Applications on Heterogeneous Systems, GPGPU 16 Proc. 9th Annu. Work. Gen. Purp. Process. using Graph. Process. Unit, 2016.

[21] V. T. Ravi, M. Becchi, W. Jiang, G. Agrawal, and S. Chakradhar, Scheduling concurrent applications on a cluster of CPU-GPU nodes, Futur. Gener. Comput. Syst., vol. 29, no. 8, pp. 22612271, 2013.

[22] B. Taylor, V. S. Marco, and Z. Wang, Adaptive optimization for OpenCL programs on embedded heterogeneous systems, Proc. 18th ACM SIGPLAN/SIGBED Conf. Lang. Compil. Tools Embed. Syst. - LCTES 2017, pp. 1120, 2017.

[23] O. E. Albayrak, I. Akturk, and O. Ozturk, Effective kernel mapping for OpenCL applications in heterogeneous platforms, Proc. Int. Conf. Parallel Process. Work., pp. 8188, 2012.

[24] Binotto, A.P., Pereira, C.E., Kuijper, A., Stork, A. and Fellner, D.W., 2011, September. An effective dynamic scheduling runtime and tuning system for heterogeneous multi and many-core desktop platforms. In High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on (pp. 78-85). IEEE.

[25] C. Gregg, M. Boyer, K. Hazelwood, and K. Skadron, Dynamic Heterogeneous Scheduling Decisions Using Historical Runtime Data, Work. Appl. Multi-and Many-Core Process., 2011.

[26] R. Kaleem, R. Barik, T. Shpeisman, B. T. Lewis, C. Hu, and K. Pingali, Adaptive heterogeneous scheduling for integrated GPUs, Proc. 23rd Int. Conf. Parallel Archit. Compil. - PACT 14, pp. 151162, 2014.

[27] Y. Wen, Z. Wang, and M. F. P. OBoyle, Smart multi-task scheduling for Open CL programs on CPU/GPU heterogeneous platforms, 2014 21st Int. Conf. High Perform. Comput. HiPC 2014, 2014.

[28] Ray, Partha Pratim. "The green grid saga-a green initiative to data centers: a review." arXiv preprint arXiv:1208.0593 (2012).

[29] T. D. Braun et al., A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, J. Parallel Distrib. Comput., vol. 61, no. 6, pp. 810837, 2001.

[30] A. Aditya, U. Chatterjee, and S. Gupta, A Comparative Study of Different Static and Dynamic Load Balancing Algorithm in Cloud Computing with Special Emphasis on Time Factor, Int. J. Curr. Eng. Technol., vol. 5, no. 3, pp. 18981907, 2015.

[31] Sanders J, Kandrot E. CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional; 2010 Jul 29.

[32] A. Tchernykh et al., Online Bi-Objective Scheduling for IaaS Clouds Ensuring Quality of Service, J. Grid Comput., vol. 14, no. 1, pp. 522, 2016.

[33] Grewe, D., Wang, Z. and OBoyle, M.F., 2013, September. OpenCL task partitioning in the presence of GPU contention. In International Workshop on Languages and Compilers for Parallel Computing (pp. 87-101). Springer, Cham.

[34] Cummins, C., Petoumenos, P., Wang, Z. and Leather, H., 2017, February. Synthesizing benchmarks for predictive modeling. In Code Generation and

Optimization (CGO), 2017 IEEE/ACM International Symposium on (pp. 86-99). IEEE.

[35] Hussain, A., Aleem, M., Khan, A., Iqbal, M.A. and Islam, M.A., RALBA: a computation-aware load balancing scheduler for cloud computing. Cluster Computing, vol. 6, pp. 114, 2018.

[36] S. K. Panda and P. K. Jana, SLA-based task scheduling algorithms for heterogeneous multi-cloud environment, J. Supercomput., 2017.

[37] T. N. Phyu, Survey of Classification Techniques in Data Mining, Int. Multi-Conference Eng. Comput. Sci., vol. I, pp. 1820, 2009.

[38] S. Gupta, A Regression Modeling Technique on Data Mining, vol. 116, no. 9, pp. 2729, 2015.

[39] Y. Chen and R. H. Katz, Analysis and Lessons from a Publicly Available Google Cluster Trace, System, p. 11, 2010.

[40] S. Kavulya, J. Tany, R. Gandhi, and P. Narasimhan, An analysis of traces from a production MapReduce cluster, CCGrid 2010 - 10th IEEE/ACM Int. Conf. Clust. Cloud, Grid Comput., no. December, pp. 94103, 2010.

[41] Z. Liu and S. Cho, Characterizing machines and workloads on a Google cluster, Proc. Int. Conf. Parallel Process. Work., pp. 397403, 2012.

[42] Baltruaitis, T., Ahuja, C. and Morency, L.P., 2018. Multimodal machine learning: A survey and taxonomy. IEEE Transactions on Pattern Analysis and Machine Intelligence.

[43] Liaw and M. Wiener, Classification and Regression by randomForest, R news, vol. 2, no. December, pp. 1822, 2002.

[44] Chen, T., He, T. and Benesty, M., 2015. Xgboost: extreme gradient boosting. R package version 0.4-2, pp.1-4.

[45] Viola, P. and Jones, M., 2001. Rapid object detection using a boosted cascade of simple features. In Computer Vision and Pattern Recognition, 2001. CVPR

2001. Proceedings of the 2001 IEEE Computer Society Conference on (Vol. 1, pp. I-I). IEEE.

[46] V. W. Lee and Others, Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU, Isca, vol. 38, no. 3, pp. 451460, 2010.

[47] B. A. Hechtman and D. J. Sorin, Exploring Memory Consistency for Massively-Threaded Throughput-Oriented Processors, ACM SIGARCH Comput. Archit. News, vol. 41, no. 3, pp. 201212, 2013.

[48] S.-Y. Lee and C.-J. Wu, Performance Characterization, Prediction, and Optimization for Heterogeneous Systems with Multi-Level Memory Interference, in 2017 IEEE International Symposium on Workload Characterization (IISWC), 2017, pp. 4353.

[49] Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. 61(6), 810837 (2001)

[50] Aditya, A., Chatterjee, U., Gupta, S.: A comparative study of different static and dynamic load balancing algorithm in cloud computing with special emphasis on time factor. Int. J. Curr. Eng. Technol. 5(3), 22774106 (2015)

[51] Elzeki, O.M., Rashad, M.Z., Elsoud, M.A.: Overview of scheduling tasks in distributed computing systems. Int. J. Soft Comput. Eng. 2(3), 470475 (2012)

[52] Li, B., Pei, Y., Wu, H., Shen, B.: Heuristics to allocate highperformance cloudlets for computation offloading in mobile ad hoc clouds. J. Supercomput. 71(8), 30093036 (2015)

[53] Biradar, S., Pawar, D.: A review paper of improving task division assignment using heuristics. Int. J. Sci. Res. 4(1), 609613 (2015)

[54] Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto Cluster Computing heterogeneous computing systems. J. Parallel Distrib. Comput. 59(2), 107131 (1999)

[55] Tchernykh, A., et al.: Online Bi-Objective Scheduling for IaaS Clouds Ensuring Quality of Service. J. Grid Comput. 14(1), 522 (2016)

[56] Hung, T.C., Phi, N.X.: Study the effect of parameters to load balancing in cloud computing. Int. J. Comput. Netw. Commun. 8(3), 3345 (2016)

[57] Maipan-uku, J.Y., Muhammed, A., Abdullah, A. and Hussin, M., 2016. Max-Average: An Extended Max-Min Scheduling Algorithm for Grid Computing Environtment. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 8(6), pp.43-47.

[58] Parsa, S., Entezari-Maleki, R.: RASA: a new grid task scheduling algorithm. Int. J. Digit. Content Technol. Appl. 3(4), 152160 (2009)

[59] Yu, X., Yu, X.: A new grid computation-based minmin algorithm. In: Sixth International Conference on Fuzzy Systems and Knowledge Discovery, pp. 4345 (2009)

[60] Panda, S.K., Agrawal, P., Khilar, P.M., Mohapatra, D.P.: Skewness-based minmin maxmin heuristic for grid task scheduling. In: Proceedings of the 2014 Fourth International Conference on Advanced Computing and Communication Technologies, pp. 282289 (2014)

[61] Long, S. and O'Boyle, M., 2004, June. Adaptive java optimisation using instance-based learning. In Proceedings of the 18th annual international conference on Supercomputing (pp. 237-246). ACM.

[62] Mohialdeen, I.A.: Comparative study of scheduling algorithms in cloud computing environment. J. Comput. Sci. 9(2), 252263 (2013)

[63] Mathew, T.: Study and analysis of various task scheduling algorithms in the cloud computing environment. In: IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 658664 (2014)

[64] Dehkordi, S.T., Bardsiri, V.K.: TASA: a new task scheduling algorithm in cloud computing. J. Adv. Comput. Eng. Technol. 1(4), 2532 (2015)

[65] Lattner, C., 2008, May. LLVM and Clang: Next generation compiler technology. In The BSD conference (pp. 1-2).

[66] Khalid, Y.N., Aleem, M., Prodan, R., Iqbal, M.A. and Islam, M.A., 2018. E-OSched: a load balancing scheduler for heterogeneous multicores. The Journal of Supercomputing, pp.1-33.

[67] Ravi, V.T., Becchi, M., Jiang, W., Agrawal, G. and Chakradhar, S., 2012, May. Scheduling concurrent applications on a cluster of cpu-gpu nodes. In Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on (pp. 140-147). IEEE.

[68] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

[69] Braun, T.D., et al.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. 61(6),810837 (2001)