RESOURCE PERFORMANCE AND QoS GUIDED INDEPENDENT TASK SCHEDULING IN GRID COMPUTING

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

CHAUHAN SAMEERSINGH ASHOKSINGH

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)

JUNE, 2010

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled **"RESOURCE PERFORMANCE AND QoS GUIDED INDEPENDENT TASK SCHEDULING IN GRID COMPUTING"** towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted to the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India is an authentic record of my own work carried out during the period from July 2009 to June 2010, under the guidance of **Dr. R. C. Joshi, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.**

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: June 14, 2010
Place: Roorkee.

(CHAUHAN SAMEERSINGH ASHOKSINGH)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: June 14, 2010.
Place: Roorkee.

(Dr. R. C. JOSHI)
Professor
Department of Electronics and Computer Engineering
Indian Institute of Technology Roorkee

Acknowledgement

It gives me immense pleasure to express my deepest sense of gratitude towards my guide **Dr. R. C. Joshi,** Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for his expert guidance, encouragement and support throughout the dissertation work. His suggestions and invaluable ideas provided the platform to the entire dissertation work. In spite of his extremely busy schedule, I have always found him accessible for suggestions and discussions. I look at him with great respect for his profound knowledge and relentless pursuit for perfection. His ever-encouraging attitude and help has been immensely valuable.

Nothing would have been possible without the support of my family members, who have been backing me up throughout my life. I wish to convey my sincere thanks to my parents, **Ashok Singh Chauhan and Shanti Devi**. I also wish to convey my sincere gratitude to my wife, **Manjari** and son, **Avi**, who have given me endless support and love. Without their support, it would not be possible to reach this far with my studies.

I would also like to thank all faculty members of Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee for their kind help and support.

CHAUHAN SAMEERSINGH ASHOKSINGH

Abstract

Task scheduling in grid computing is a challenging problem because of heterogeneous and dynamic nature of grid resources. The performance of grid resources is constantly changing. Task scheduling becomes more complicated when various quality of service (QoS) demands arise from users. QoS is an extensive concept and it varies from application to application. QoS is set of constraints for effective execution of an application.

In this dissertation work, six heuristics are proposed for independent task batch mode scheduling. Two heuristics, namely, segmented weighted time-min (SWT-Min) and segmented weighted time-max (SWT-Max) are for resource performance based independent task scheduling. Both heuristics first finds the performance metric of resources and uses this metric to find the weighted time of each task. They divide the tasks in number of segments. Each segment is assigned a priority and mapping of tasks is done in the descending order of priority of segments. Four heuristics, namely, QoS guided weighted mean time-min(QWMTM), QoS guided weighted mean time min-min max-min selective(QWMTS), priority based QoS guided weighted time min-min max-min selective(PQWMTS) and multiple QoS guided weighted mean time min-min max-min selective(MQWMTS) are proposed for QoS based independent task scheduling. The QWMTM, QWMTS and PQWMTS heuristics are for single QoS based task scheduling and network bandwidth is considered as QoS parameter. The MQWMTS heuristic is for multiple QoS based task scheduling. Response time, execution cost and priority are considered as QoS parameters for testing the heuristic. A generalized function is presented to consider all QoS and to generate a utility value of tasks. This utility value decides the order of execution of tasks.

The gridsim simulation toolkit is used to validate the proposed heuristics. The heuristics are evaluated on the basis of makespan and resource load balancing. The results show that all proposed heuristics gives good improvements in makespan and do better resource load balancing than existing heuristics such as QoS guided min-min, weighted mean time-min, weighted mean time min-min max-min selective, min-min and max-min.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction and Problem Statement

1.1 Introduction

The rapid development of Internet technologies, advances in wide-area network technologies and low cost computing have given birth to a new computing era known as Grid computing. Grid is a collection of wide variety of resources such as computer systems, storage spaces, specialized devices, software applications etc. A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities[1]. The grid resources together create tremendous processing power. Efficient and effective scheduling algorithms are needed to fully utilize the computational power of grid resources.

Task scheduling is one of the integral components of grid computing. It is a process of mapping a set of tasks on a set of machines. Effective task scheduling is the key for high performance in grid computing. It is a challenging problem because of heterogeneity and dynamic nature of grid resources. The grid resources are owned by multiple organizations and governed by various user and system access policies. The access policies decide who will access what, how and when. It has been proved that task scheduling in heterogeneous environment like grid is a NP-hard[2] problem. Many heuristic algorithms such as Min-Min[3], Max-Min[3], Sufferage[3], X-Sufferage[4] are proposed to find the optimal solution. These algorithms, depending on goals, assign tasks to the best machine which produces better quality of service.

QoS is one of the crucial means to satisfy the various demands from resource users and resource providers. It is a very extensive concept and it varies from application to application. It can be network bandwidth, execution cost, reliability, availability, trust etc. The classical scheduling algorithms such as Min-Min, Max-Min, Sufferage, etc. do not consider influence brought by QoS. Many heuristics such as QoS guided Min-Min[5], QoS priority grouping[6], etc. have been proposed for QoS based task

scheduling. Results given by these QoS based scheduling algorithms are far better than that of classical scheduling algorithms.

Makespan is the standard metric for performance measure of a scheduling algorithm in grid. Makespan for a schedule can be defined as the maximum time taken by a task to complete its execution. The shorter the makespan, the better the scheduling algorithm is. Both user and system want to complete the task as soon as possible. So, one of the primary goals of any scheduling algorithm should be to reduce the makespan. Load balancing is a vital problem in grid computing. The factors such as dynamicity, heterogeneity, multiple administrative policies etc., affect to balance the load across resources. The resources are loaded with their own internal loads, so the response time cannot be guaranteed.

1.2 Motivation

Grid provides a platform for sharing, selection, and aggregation of geographically distributed heterogeneous resources. Users can submit their applications from anywhere to grid. The applications are of different types and they require different types of resources to execute them. The applications demand various types of QoS to complete their execution as fast as possible. The various QoS demands can be network bandwidth, execution cost, reliability, availability, trust, etc. Users wish to get the results as early as possible. So, one of the problems of scheduling algorithms is how to reduce makespan. Every user tries to get the best resources for his application. This creates the resource load unbalancing. So, another problem of scheduling algorithm is how to balance the load across resources. To reduce the makespan and balance the load across resources have motivated to design and propose new scheduling heuristics.

1.3 Problem Statement

The main objective of this dissertation work is to propose resource performance and QoS guided independent task scheduling algorithms for grid computing.

To achieve the main objective, it is further divided in following sub-objectives:

i) To propose heuristics based on the performance of resources.

ii) To propose heuristics for single QoS constraint based scheduling.

iii) To propose heuristics for multiple QoS constraint based scheduling.

iv) To validate the proposed heuristics.

1.4 Organization of Report

The dissertation report is organized in 7 chapters including this chapter. This chapter gives introduction and states the problem. The rest of report is organized as follows.

Chapter 2, overview of grid computing, task scheduling in grid and QoS in grid in discussed. The research gaps and literature review is also given in the chapter.

Resource performance based independent task scheduling heuristics are presented in chapter 3. Two heuristics, namely, Segmented Weighted Time-Min and Segmented Weighted Time-Max are proposed for resource performance based task scheduling. Both heuristics, first, find the performance metric of each resource. This performance metric is used to assign a weight value to each resource. The segmentation of tasks is done to allow the long tasks to be executed first.

QoS guided independent task scheduling heuristics are presented in chapter 4. Total four heuristics are presented, three are for single QoS constraint based scheduling and one is for multiple QoS constraints based scheduling.

In Chapter 5 simulation tool, simulation environment and the performance metrics are discussed. The gridsim simulation toolkit is used for simulation purpose. The performance metrics, makespan and load balance degree are used to validate the proposed heuristics.

Results are discussed in chapter 6. Chapter 7 concludes the dissertation and discusses the scope for future work.

Chapter 2

Background and Literature Review

2.1 Grid Computing - Overview

The rapid development of Internet technologies, advances in wide-area network technologies and low cost computing have given birth to a new era known as Grid computing. Grid is a collection of wide variety of resources including computer systems, storage spaces, specialized devices, software applications which are geographically distributed and owned by multiple organizations. The access of these resources is governed by organizations' individual administrative policies. Grids enable sharing, selection and aggregation of resources for solving large-scale computational and data intensive problems in science, engineering, and commerce. In Grid computing, computing becomes pervasive and individual users or client applications gain access of computing resources as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on are. Grid is "*A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost and users' quality-of-service requirements*"[7].

2.1.1 Grid Systems

The business problems and applications have diverse need of computational resources. Some applications need high computational power, some needs specific service and so on. Based on the business problem or application requirements, the type of a grid is selected. According to the distinct targeted applications domains, a grid can be classified in to three categories[8]: computational grid, data grid and service grid, as shown in figure 2.1. Real grid can be a combination of two or more of these types. Development of truly general-purpose grid that can support multiple or all of these categories is a hard problem.

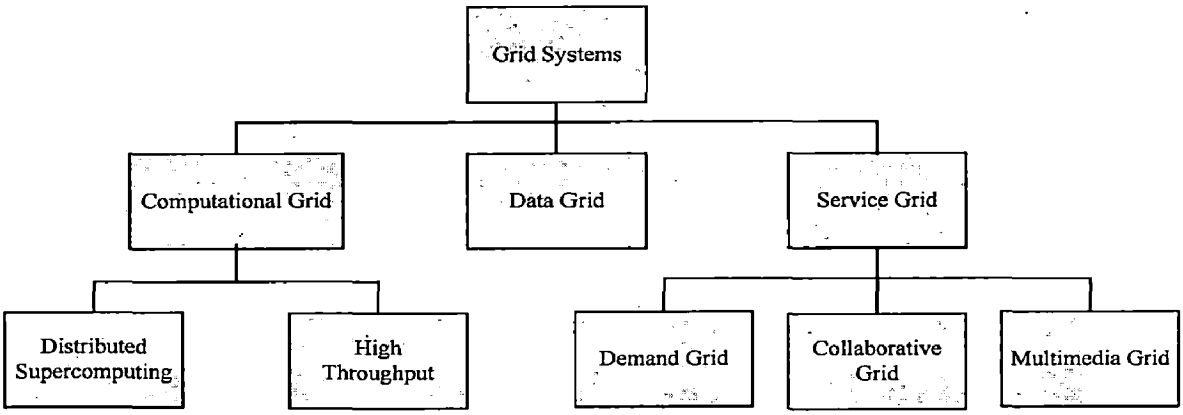


Figure 2.1 Grid Systems

i) **Computational Grid:** -A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities[1]. A computational grid aggregates the processing power of geographically distributed heterogeneous systems. These distributed systems, together, have a higher aggregate computational capacity than the capacity of any constituent machine in the system. According to how the computational power is utilized, computational grid can be further classified into distributed supercomputing and high throughput. A distributed supercomputing grid executes the applications in parallel on multiple machines to reduce the completion time of a job. A high throughput grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as many as possible.

ii) **Data Grid:** - A grid providing an integrated view of data storage is called a "data Grid"[8]. A data grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned where data are located as far as they have access to the data.

iii) **Service Grid:** - A service grid[8] provides the services those are not provided by any single machine. This type of grid is further classified as on demand, collaborative and multimedia grid. An on demand grid dynamically aggregates different resources to provide new services. Collaborative grid provides real time interaction between users and applications via a virtual workspace by connecting users and applications into collaborative

environment. A multimedia grid provides an infrastructure for real time multimedia applications.

2.2 Task Scheduling in Grid

Task scheduling is a process of mapping a set of tasks on a set of machines. Effective task scheduling is the key for high performance in grid computing. A generalized grid scheduling system is shown in figure 2.2. There are mainly four modules namely, grid scheduler, grid information service, launching and monitoring and local resource manager. The broken lines in figure show resource or application information flows and solid lines show task or task scheduling information flows.

i) **Grid Scheduler:** - The grid users submit their jobs or applications to grid scheduler. The grid scheduler collects information about resources from grid information service and generates application and resource mapping based on some objective specified by the user or predicted resource performance.

ii) **Grid Information Service:** - The grid information service (GIS) collects information of available resources, continuously. The GIS at equal time interval gathers this information. GIS is responsible for collecting and predicting the resource state information such as CPU capacities, memory size, network bandwidth, software availabilities and load of a site at particular period. Application properties and performance of a resource for various applications are necessary for feasible application and resource mapping. The performance model such as cost estimation helps the scheduler to choose the best match to optimize the objective functions.

iii) **Launching and Monitoring Module:** - The launching and monitoring module implements the finally determined schedule by submitting the application to selected resources and monitors the execution of applications.

iv) **Local Resource Manager:** - The local resource manager is mainly responsible for two jobs: local scheduling and reporting resource information to GIS. Local scheduling deals with scheduling of jobs of both exterior grid users and local users. There may be one or more than one local scheduler. It depends on the size, number and types of resources, and number and types of applications submitted. The local resource manager also collects information

regarding resource load, availability of resources, etc. and submits this information to GIS.

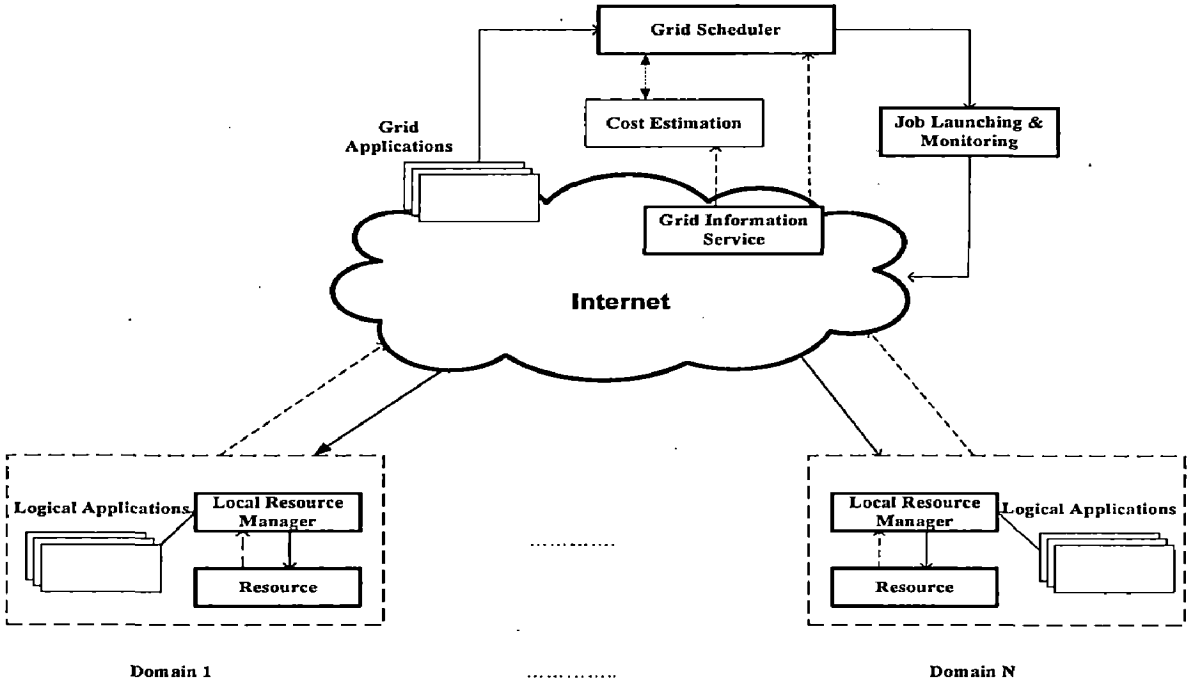


Figure 2.2 A Logical Grid Task Scheduling Architecture[9]

2.2.1 Task Scheduling Phases

The task scheduling in grid comprises three phases.

i) **Resource Discovery: -** In resource discovery[10] phase, a list of potential resources is generated. These resources are investigated in more detail in phase two. The resource discovery involves three sub stages, authorization filtering, application definition and minimal requirement filtering. In authorization filtering, the resources are discovered and determined whether they are accessible or not. In grid, resources are monitored by multiple administrative domains and the scheduler has to follow all administrative policies to have the access of resources from different domains. At the end of authorization filtering, user have a set of resources on which access have been granted. In application definition, the user should be able to specify some minimal set of job requirements. This will help in further filtering of resources. The set of possible job requirements may vary from user to user

and application to application. Some requirements may be static like operating system and some may be dynamic such as RAM requirements, bandwidth, or connectivity. In minimal requirement filtering stage, the resources are filtered which do not fulfill the minimal job requirements. As the grid resources are distributed and dynamic in nature, this step uses static data to evaluate whether a resource meets the minimal requirements. The resources are dropped which do not satisfy the need for job requirement. At the end of this stage, a set of resources is available for further investigation.

ii) **Resource Selection:** - In resource selection[10] phase the given a set of resources, generated in first phase, are further investigated and a resource or a set of resources is selected for job execution. In information gathering stage, detailed information about the resources is collected. This may include the availability duration, reliability, access rights, etc. This information is gathered dynamically. The gathered information will be consist or not, it is hard to tell, because resources are dynamic in nature and governed by multiple administrative policies. Many system gather information periodically. In system selection stage, a resource or set of resources is selected for job execution. This selection can be the best resource from the set of resources or based on other issues like availability, reliability etc.

iii) **Job Execution:** - The last phase is job execution[10]. This may involve many sub stages. First sub-stage is to do advance reservation. Advance reservation of resources is good because it guarantees that resources will be available at the time of job execution. Once resources are chosen, the next stage is to submit the job. It can be very simple to run just one command or very complicated as to run series of commands. The preparation stage may involve setup, staging, claiming a reservation or other actions needed to prepare the resource to run job. Next stage is monitoring the progress. It depends on the type of application. User can decide how to monitor the progress. It can send a query at regular time interval to the resource for status of application. When job finishes the user is notified and the results are given. The last stage is clean up tasks; it involves the retrieving of files from the resources and to do analysis on results. It also involves removing temporary settings.

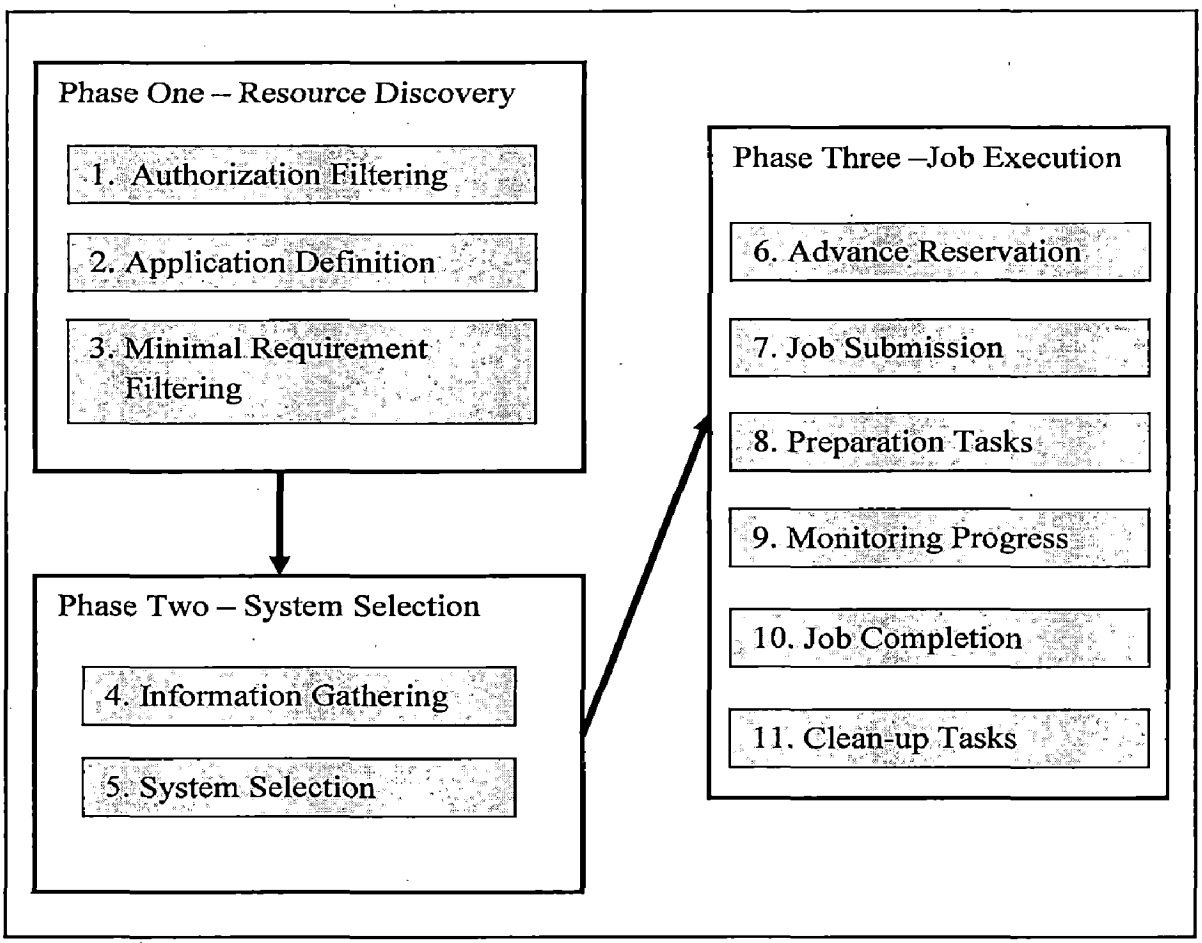


Figure 2.3 Task Scheduling Phases[10]

2.2.2 Challenges of Task Scheduling in Grid

Grid is having some unique characteristics, explained below. These characteristics make the task scheduling a challenging problem.

i) **Heterogeneity:** - In grid, resources are distributed in multiple administrative domains. The resources are of wide variety and are having varying characteristics. The computational, storage and underlying connecting networks are heterogeneous. The heterogeneity results in different capabilities for job processing and data access.

ii) **Autonomy:** - In grid resources are owned by various organizations. They are having different access and security policies. Grid does not have full control over the resources. Schedulers have to work with the local policies of resources, which makes it hard to estimate the exact cost of executing a task

on different sites. The autonomy results in the diversity in local resource management and access control policies. A grid scheduler is required to be adaptive to different local policies.

iii) **Performance Dynamism:** - Grid is dynamic and service providers leave and join it based on their feasibility. Because of the dynamic nature of grid resources, the performance is constantly changing and creates performance dynamism[9]. This makes the grid schedulers to work in dynamic environment. Because of local autonomy of resources, they are not dedicated to grid applications. For example, a grid job submitted remotely to a computer cluster might be interrupted by a cluster's internal job which has a higher priority. The network bandwidth is heavily affected by Internet traffic that is not relevant to grid applications. This type of contention leads to performance fluctuation to the grid applications. It makes a hard job to evaluate the grid scheduling performance under classical performance model. A feasible scheduling algorithm should be able to adapt such dynamic behaviors.

iv) **Resource Selection:-** A grid is having a large number of computational and storage resources, distributed in multiple administrative domains. Grid scheduler selects computational nodes based on resource model and performance criteria. The communication bandwidth in grid is shared by background loads, so inter-domain communication cost should be considered. Many grid applications are data intensive, so data staging cost is also considerable. These all factors make difficult to select a resource for computation because if one resource is providing low computational cost may require high access cost to storage site.

2.2.3 Mapping Modes

Tasks can be divided into two groups: Independent and Dependent[11]. Dependent task depends on the results of its predecessors while independent task do not require communication with other tasks in same metatask.[12]. Metatask is a collection of independent tasks with no inter-task data dependencies. Mapping of tasks can be done in two modes: On line mode and batch mode.

i) **On line mode:** - Task is mapped to a resource as soon as it arrives to the scheduler. Task is considered only once for mapping and the mapping does not change once it is done. This mode of mapping is useful when task arrival rate is low.

ii) **Batch mode:** - In this mode, first, tasks are collected into a set. This set is called metatask. Mapping of metatask is performed at prescheduled time called mapping events. Mapping of each task is performed at every mapping event until it begins its execution. This mode can make better decisions because heuristics have the resource requirement information of the metatask and the actual execution time of a larger number of tasks is also known in advance.

2.3 QoS in Grid

In grid, the quality of services (QoS) is a big concern for many applications. It is an extensive concept which varies from application to application. It could be the requirement of CPU speed, network bandwidth, deadline, execution cost etc. QoS is a set of conditions to run an application. Providing nontrivial QoS is one of the primary goals of grid computing. Based on the grid infrastructure two types of QoS attributes can be distinguished: Qualitative and Quantitative. The qualitative QoS attributes refers to user satisfaction and service reliability while the quantitative QoS attributes refers to network latency, CPU performance, storage capacity, etc. For example, parameters for network QoS can be bandwidth, delay, throughput, parameters for CPU computing can be specified based on how the CPU is used. CPU can be used either in shared mode or in exclusive mode. In shared mode the user application shares the CPU processing power with other user applications while in exclusive mode user application has complete access of the CPU. Parameters for storage QoS are bandwidth and storage capacity. Users can have many types of QoS requirements for their applications. It can be single or multiple QoS. Some possible user requirements are:

i) **Timelineness:** - Timelineness[13] defines the requirement of time associated with a task or application. It can be total execution time, deadline or response time etc.

ii) **Execution Cost:** - Execution cost refers to the cost paid by the user to the service provider to execute the application.

iii) **Priority:** - Each user can assign priority[13] to his tasks. The higher priority tasks can be schedule by scheduler with higher importance.

iv) **Reliability:** - The task that runs for a long time can experience failure during execution. The reliability of a task is defined to be the probability that task can be completed successfully[14]. Each user may specify a degree of reliability for its tasks.

2.3.1 Layered Structure of Grid QoS

The layered structure of grid QoS[15] is shown in figure 2.4. The top layer represents QoS demands from the user. User can have single or multiple QoS demands for his application. The second layer represents the grid service QoS. In this layer, the service providers define the specific descriptive QoS parameters such as service security QoS, reliability QoS, and accounting QoS. The service providers also define the simplified QoS level such as bad, general, good, better, best. This can help to meet the user's possible simple QoS demands. The third layer is for system and logical resource QoS. To meet user's demands on functional QoS parameter, the service provider should define the different specific system QoS parameters and logical resource QoS parameters. This layer's QoS parameters refer to the mapped parameters of upper layer QoS. To descriptive QoS parameters, each sub service can translate them into their own security QoS, reliability QoS and accounting QoS based on their semantic. To functional QoS parameters, the service provider should define each sub service different specific system QoS parameters and logical resource QoS parameters, which corresponds to different simplified QoS levels in upper layer. The bottom layer is for physical parameter such as network QoS, device QoS, and physical resource QoS. The QoS parameters in this layer are translated QoS parameters from the upper layers. System QoS is translated into device and network QoS, logical resource QoS is translated into physical resource QoS.

The bottom layer consists of network QoS, device QoS and physical resource QoS is transparent to user. So, the user can put its demand either in grid service layer or system or logical resource layer. In grid service layer user can put demands for QoS

like good, best etc. In system or logical resource QoS layer the user can put specific demands like system response time <=120 seconds or transmission speed should be >= 128 Kbps. Because the relationship of QoS parameters between different layers is defined by the service provider, the provider should define the specific QoS parameters in each layer.

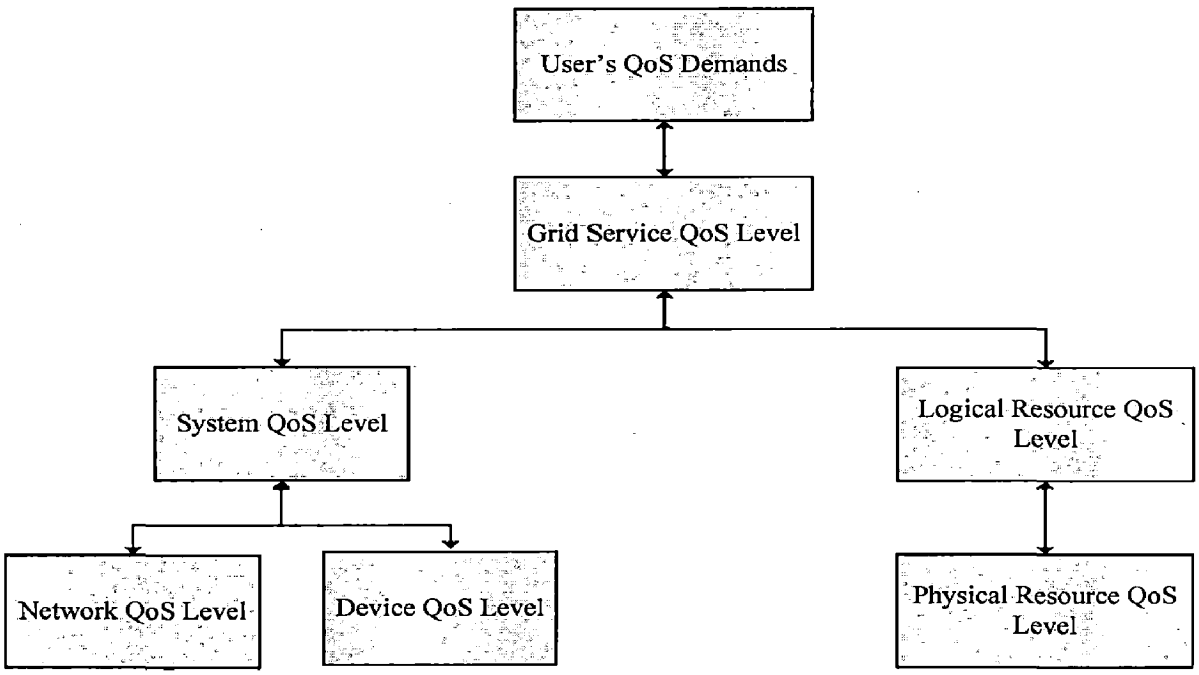


Figure 2.4 Layered Structure of Grid QoS[15]

2.4 Literature Review

In this section, two types, batch mode and QoS based batch mode scheduling heuristics for independent task is discussed.

2.4.1 Batch Mode Heuristics

i) **Min-Min:** - The Min-Min[2] heuristic is a simple algorithm which runs fast and delivers the satisfactory performance. Min-Min begins with the set MT of all unassigned tasks. It has two phases. In the first phase, the set of minimum expected completion time for each task in MT is found. In the second phase, the task with the overall minimum expected completion time from MT is chosen and assigned to the corresponding resource. Then this mapped task is

removed from MT and the process is repeated until all tasks in the MT are mapped. Min-Min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Let t_i be the first task mapped by min-min onto an empty system. The machine that finishes t_i the earliest, say m_j, is also the machine that executes t_i the fastest. For every task that min-min maps after t_i, it changes the availability status of m_j by the least possible amount for every assignment. In most situations, it maps as many tasks as possible to their first choice of service resources. In min-min, it is expected that a smaller makespan can be obtained if more tasks are assigned to the machines that completes them the earliest and also executes them fastest. However, the Min-Min algorithm is unable to balance the load well since it usually schedules small tasks first.

ii) **Max-Min: -** Max-Min[2] heuristic is very similar to Min-Min, except in second phase. Max-Min assigns task with maximum expected completion time to the corresponding resource in second phase. The Max-Min algorithm may give a mapping with more balanced loads across the service resources in some environments. Max-Min attempts to minimize the penalties incurred from performing tasks with longer execution times. For example, let there are many tasks with shorter execution times and one task with larger execution time. Mapping the task with larger execution time to its best machine allows this task to be executed concurrently with the remaining tasks, having shorter execution time. In this case the max-min will give better mapping than min-min by executing larger task with parallel shorter tasks. In cases similar to this example, the max-min heuristic may give more balanced load and better makespan.

iii) **Weighted Mean Time-Min: -** Weighted Mean Time-min heuristic[16] employs weighted mean execution time as heuristic and then assigns the tasks which is having maximum weighted mean execution time to the machine with minimum earliest completion time. The heuristic finds the performance metric of each resource, called the weight of the resource. This weight is used to find the weighted mean time of each task.

iv) **Segmented Min-Min:** - In Segmented Min-Min heuristic described in [17] tasks are first ordered by their expected completion times. Then the ordered sequence is segmented and finally it applies Min-Min to these segments. This heuristic works better than Min-Min when length of tasks are dramatically different by giving a chance to longer tasks to be executed earlier than where the original Min-Min is adopted.

v) **Sufferage:** - The Sufferage[3] heuristic is based on the idea that better mapping can be generated by assigning a machine to a task that would suffer most in terms of expected completion time if that particular machine is not assigned to it. For each task, its sufferage value is defined as the difference between its best minimum completion time and its second best minimum completion time. Tasks with high sufferage value take precedence. But when there is input and output data for the tasks, and resources are clustered, sufferage heuristic may have problems. In this case, intuitively, tasks should be assigned to the resources as near as possible to the data source to reduce the makespan. But if the resources are clustered and nodes in the same cluster are with near identical performance, then the best and second best minimum are also nearly identical which makes sufferage value close to zero and gives the tasks low priority.

vi) **XSufferage:** - XSufferage[4] heuristic is proposed to solve the problem of conventional sufferage heuristic. It gives cluster level sufferage value in place of total resource level as given by sufferage heuristic. Experiments show that XSufferge heuristic outperforms the conventional sufferage heuristic.

2.4.2 QoS Based Heuristics

i) **QoS Guided Min-Min:** - QoS Guided Min-Min heuristic[5] is based on the Min-Min heuristic. It considers network bandwidth as QoS parameter. It divides the tasks in two groups: high and low QoS. The idea behind this division is that the tasks requiring high QoS can only run on high QoS providing hosts. The low QoS task can run on any hosts and if they are allocated to high QoS resources, then it leads large makespan, wastage of resources and unbalancing the load. At last, this reduces the overall performance of grid systems. The QoS guided Min-Min heuristic first

schedules the tasks from high QoS group on resources that can provide high QoS as required. Later it schedules the tasks from low QoS group

ii) **Priority Grouping Heuristic:** - The priority grouping[6] algorithm creates priority groups. These groups are created on the basis of the QoS services provided by the resources. If there are n resources and each one can provide different QoS services, then n groups can be created. Each group is assigned a priority level. The task is assigned to one of these groups based on the QoS requirement of it. In the descendent order from high to low priority, the tasks from different groups are scheduled.

2.5 Research Gaps

There are many heuristics such as given in section 2.4.1 and 2.4.2 for batch mode and QoS based independent task scheduling. There is scope to improve the existing heuristics as well as propose new heuristics. Combination of more than two heuristics gives better results than single heuristic. There are no such heuristic that should consider resource performance as well as QoS requirements of tasks for mapping. The heuristic based on resource performance and QoS demands can be proposed.

2.6 Terminology

The following terminology, give in [5], is used throughout the report.

i) **Expected Execution Time:** - The expected execution time ET_{ij} of task t_i on resource r_j is defined as the time taken by resource r_j to execute the task t_i, when there is no load assigned to resource r_j.

ii) **Expected Completion Time:** - The expected completion time CT_{ij} of task t_i is defined as the time taken by resource r_j to complete the execution of task t_i, after finishing the previously assigned load. Hence,

$$CT_{ij} = ET_{ij} + rt_j \tag{2.1}$$

Here, rt_j is the ready time or time needed to complete the previous assigned load. The above mentioned terminology can be obtained though predication model.

Chapter 3

Proposed Resource Performance Guided Independent Task Scheduling Heuristics

Resource performance refers to the performance metric of a resource in a mapping schedule. The performance metric of each resource is computed and a weight value is assigned to each resource based on the performance metric. This weight value is used to compute the weighted expected execution time. Two heuristics, namely, segmented weighted time – min and segmented weighted time – max are based on this performance metric. Both heuristics divides the tasks in number of segments on the basis of minimum weighted expected execution time. Each segment is assigned a priority. In descending order of priority of segments, the tasks from segments are mapped. In both heuristics total number of tasks in metatask is referred with n and total number of resources is referred with m. The heuristics segmented weighted time – min and segmented weighted time – max are described in section 3.1 and section 3.2, respectively.

3.1 Segmented Weighted Time-Min Heuristic

The segmented weighted time-min (SWT-Min) heuristic is shown in figure 3.1. It is assumed that the expected execution time of all tasks on each resource is known in advance. The steps are elaborated in detail below.

a) **Find the performance metric of each resource:** - The method given in [16], is used to find the performance metric of each resource. First average execution time of all tasks on each resource is found. It can be computed using the equation (3.1).

$$avg_j = \frac{\sum_{i=1}^{n} ET_{ij}}{n} \tag{3.1}$$

Here, ET_{ij} is the expected execution time of task t_i on resource r_j. This average execution time can be used to represent the performance of resource. If $avg_i < avg_j$, the performance of resource r_i is better than resource r_j. The average

execution time values can be used to find the weight, the performance metric, of each resource. It can be computed using equation (3.2).

$$w_i = \frac{avg_i}{\sum_{j=1}^{m} avg_j} \tag{3.2}$$

The smaller the value of w_j, the better the resource is and

$$\sum_{i=1}^{m} w_i = 1 \tag{3.3}$$

b) **Find weighted expected execution time of each task: -** The weighted expected execution time of each task is computed using equation (3.4).

$$WET_{ij} = w_j \times ET_{ij} \tag{3.4}$$

c) **Create Segments: -** The minimum weighted expected execution time of each task is found. The tasks are sorted on the basis of minimum weighted time. Depending of the number of tasks and number of resources, segments of tasks is created. Each segment is assigned equal number of tasks. A priority value is assigned to each segment. The segment having tasks with smaller weighted expected execution time is assigned lowest priority and the segment having tasks with larger weighted time is assigned highest priority. It is difficult to decide, how many segments should be created. It all depends on the number of tasks in the batch and number of resources available at the time of scheduling. The segmentation helps to schedule the tasks having larger execution time first. It finally results in better resource load balancing.

d) **Scheduling Process: -** In the descending order of priority of segments, the tasks from segments are mapped. For each segment, the heuristic performs the following steps. The heuristic finds the task with minimum weighted time and maps it on the resource that is giving earliest completion time. If there are more than one resource which are giving earliest completion time then the resource with least load is selected. This helps in balancing the resource load. The mapped task is deleted from the metatask and ready time of the resource is updated. This process continues until all tasks from the segments are mapped.

```
(1)   Find the performance metric of each resource
(2)   Find weighted expected execution time of each task
(3)   Create N segments
(4)   In descending order of priority of segments, the
      tasks are mapped
(5)   While (i < N)
(6)     Do until all tasks from the i^th segment are
        mapped
(7)       Find the task t_i with minimum weighted expected
          execution time
(8)       Find the resource r_j that gives earliest
          completion time
(9)       If there are more than one resource which are
          giving earliest completion time then select the
          resource with least load
(10)      Map the task t_i on the resource r_j
(11)      Remove the task t_i from the meta-task
(12)      Update the ready time of resource r_j
(13)    End Do
(14)  End While
```

Figure 3.1 Segmented Weighted Time-Min (SWT-Min) Heuristic

3.2 Segmented Weighted Time-Max

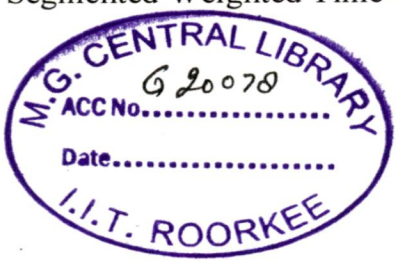
The Segmented Weighted Time-Max Heuristic is very similar to segmented weighted time-min heuristic. The heuristic is shown in figure 3.2. The working of the heuristic is very similar to segmented weighted time-min heuristic; it differs only in the scheduling process. Other process like finding the performance metric of each resource, finding weighted time and creating segments are similar to segmented weighted time-min heuristic.

a) **Scheduling Process:** - In the descending order of priority of segments, the tasks from segments are mapped. For each segment, the heuristic performs the

following steps. The heuristic finds the task with maximum weighted time and maps it on the resource that is giving earliest completion time. If there are more than one resource which are giving earliest time then the resource with least load is selected. This helps in balancing the resource load. The mapped task is deleted from the metatask and ready time of the resource is updated. This process continues until all tasks from the segments are mapped.

```
(1)   Find the performance metric of each resource
(2)   Find weighted expected execution time of each task
(3)   Create N segments
(4)   In descending order of priority of segments, the
      tasks are mapped
(5)   While (i < N)
(6)     Do until all tasks from the i^th segment are
        mapped
(7)       Find the task t_i with maximum weighted expected
          execution time
(8)       Find the resource r_j that gives earliest
          completion time
(9)       If there are more than one resource which are
          giving earliest completion time then select the
          resource with least load
(10)      Map the task t_i on the resource r_j
(11)      Remove the task t_i from the meta-task
(12)      Update the ready time of resource r_j
(13)    End Do
(14)  End While
```

Figure 3.2 Segmented Weighted Time–Max Heuristic



Chapter 4

Proposed QoS Guided Independent Task Scheduling Heuristics

QoS imposes constraints on scheduler to schedule the tasks under given conditions. The QoS demand can be for single such as network bandwidth or multiple such as cost, trust, availability, priority etc. One example can be taken to justify why QoS based task scheduling in necessary. Let there are two tasks t_1 and t_2 and two resources r_1 and r_2. Task t_1 can only be executed on resource r_1 and task t_2 can be executed on any of two resources. If task t_2 is first mapped on resource r_1, then task t_1 will wait till r_1 becomes free and at the same time resource r_2 will also be idle. This mapping will create larger makespan and resource load unbalancing. Now, if task t_1 is first mapped on resource r_1 then task t_2 can also be mapped on resource r_2 at the same time. Now no task is waiting for any resource to become free as well as all resources are engaged. This mapping reduces the makespan and balances the load across resources well. In this chapter, four QoS based heuristics are proposed. The QoS guided weighted mean time-min, QoS guided weighted mean time min-min max-min selective and priority based QoS guided weighted mean time min-min max-min selective heuristics are proposed for single QoS constraints based scheduling. Multiple QoS guided weighted mean time min-min max-min selective heuristic is proposed for multiple QoS constraints based scheduling.

4.1 QoS Guided Weighted Mean Time-Min Heuristic

The QoS guided weighted mean time-min (QWMTM) heuristic is proposed for QoS based independent task scheduling. The weighted mean time-min heuristic is modified and a QoS parameter is introduced in it. Network bandwidth is taken as QoS parameter. The heuristic is given in figure 4.1. It is assumed that the expected execution time of all tasks on each resource is known in advance. The steps are described in detail below.

i) **Divide the tasks in two QoS groups:** - The tasks are divided in two QoS groups: high and low. In high group task with high QoS demand are taken and

in low group task with low or no QoS demand are taken. The high QoS tasks can only be executed on high QoS providing resources. But the low QoS tasks can be executed on any resource. This division of tasks helps to schedule the high QoS tasks first and low QoS tasks afterward. This will also help to satisfy the QoS demands of tasks first.

ii) **Find the performance metric of resource:** - The resources are divided on the basis of their services. Two groups are created: high and low. The performance metric of resource of each group is computed. The method given in section 3.1 of chapter 3 is used to compute the resource performance metric.

iii) **Find the weighted mean time of each task:** - The equation (4.1) is used to find the weighted mean time of a task. The weighted mean time shows the overall execution time of a task on all resources. If $wmt_i < wmt_j$ then, task t_i requires less time to execute than task t_j.

$$wmt_i = \frac{\sum_{j=1}^{m} w_j ET_{ij}}{m} \tag{4.1}$$

iv) **Scheduling Process:** - First the tasks from high QoS group are mapped. The following steps are performed. The heuristic finds the task t_i with maximum weighted mean time. It finds the resource r_j from QoS qualified resources which gives the earliest completion for task t_i. It maps the task t_i on resource r_j. The mapped task t_i is deleted from the metatask. The ready time of resource r_j is updated. This process continues until all tasks with high QoS are mapped. After mapping all high QoS tasks, the tasks from low QoS group are mapped. From all tasks it finds task t_i with maximum weighted mean time and resource r_j which gives earliest completion time for task t_i. The task t_i is mapped on resource r_j. After mapping the task t_i is deleted from metatask. The ready time of resource r_j is updated and this process continues until all tasks are mapped.

```
(1)   Divide the tasks in two QoS groups : High and Low
(2)   Find the performance metric of each resource
(3)   Find weighted mean time of each task
(4)   While there are tasks in metatask
(5)      Do until all tasks with high QoS are mapped
(6)         Find task t_i with maximum weighted mean time
(7)         For the task t_i, find the resource r_j from QoS
            qualified resources that gives minimum
            completion time
(8)         Assign task t_i to resource r_j
(9)         Update ready time of resource r_j
(10)        Delete task t_i from metatask
(11)     End Do
(12)     Do until all tasks with low QoS are mapped
(13)        Find task t_i with maximum weighted mean time
(14)        For the task t_i find the resource r_j that
            gives the earliest completion time
(15)        Assign task t_i to resource r_j
(16)        Update ready time of resource r_j
(17)        Delete the task t_i from meatatask
(18)     End Do
(19)  End While
```

Figure 4.1 QoS Guided Weighted Mean Time-Min Heuristic

4.2 QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

The QoS guided weighted mean time min-min max-min selective (QWMTS) heuristic is the modified heuristic of weighted mean time min-min max-min selective (WMTS) heuristic[18]. The WMTS uses the merits and demerits of min-min and max-min heuristics for scheduling. A QoS parameter, network bandwidth is introduced in WMTS and new QWMTS heuristic is proposed. The QWMTS heuristic is shown in figure 4.2. It is assumed that the expected execution time of all tasks on each resource is known in advance. The steps are described in detail below:

i) **Divide the tasks in two QoS groups: -** The tasks are divided in two QoS groups: high and low. In high group task with high QoS demand are taken and in low group task with low or no QoS demand are taken. The high QoS tasks can only be executed on high QoS providing resources. But the low QoS tasks can be executed on any resource. This division of tasks helps to schedule the high QoS tasks first and low QoS tasks afterward. This will also help to satisfy the QoS demands of tasks first.

ii) **Find the performance metric of resource: -** The resources are divided on the basis of their QoS services. The two groups are created, in first group resources are taken which can satisfy the QoS demands of tasks and in second group the resources are taken which can not satisfy the QoS demand of tasks. The method given in section 3.1 of chapter 3 is used to compute the performance metric of resource of each group separately.

iii) **Scheduling Process: -** The tasks from high QoS group are mapped first. The heuristic perform the following steps until all tasks from group are mapped. It finds expected completion time of each task using equation (4.2)

$$CT_{ij} = ET_{ij} + rt_j \qquad (4.2)$$

It finds the weighted mean time of each task. It uses equation (4.1) to find the weighted mean time. It finds the standard deviation (SD)[19] of expected completion time of all unassigned tasks of metatask. It uses the equation (4.3) to find the standard deviation.

```
(1)   Divide the tasks in two QoS groups : high and low
(2)   Find the performance metric of each resource
(3)   While there are tasks in metatask
(4)     Do until all tasks with high QoS group are mapped
(5)      Find the expected completion time of each task
(6)      Find weighted mean time of each task
(7)      Compute the standard deviation SD
(8)      Compute relative standard deviation SD'
(9)      If SD' < ST then
(10)      Find the task t_i that is having minimum
          weighted mean execution time and assign it to
          the resource r_j, from the QoS qualified set,
          that is giving earliest completion time
         Else
(11)      Find the task t_i that is having maximum
          weighted mean execution time and assign it
          to the resource r_j, from the QoS qualified set,
          that is giving earliest completion time
(12)     Delete task t_i from meatatask
(13)     Update ready time of resource r_j
(14)    End Do
(15)    Do until all tasks with low QoS group are mapped
(16)     Find the expected completion time of each task
(17)     Find weighted mean time of each task
(18)     Compute the standard deviation (SD)
(19)     Compute relative standard deviation SD'
(20)     If SD' < ST then
(21)      Find the task t_i that is having minimum
          weighted mean execution time and assign it to
          the resource r_j, that is giving earliest
          completion time
(22)     Else
(23)      Find the task t_i that is having maximum
          weighted mean execution time and assign it to
          the resource, that is giving earliest
          completion time
(24)      Delete task t_i from metatask
(25)      Update ready time of resource r_j
(26)    End Do
(27)  End While
```

Figure 4.2 QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

$$SD = \sqrt{\frac{\sum_{i=1}^{n}(CT_{ij} - avgCT)^2}{n}} \tag{4.3}$$

Here, avgCT is average execution time of all unassigned tasks of metatask. It can be computed using equation (4.4)

$$avgCT = \frac{\sum_{i=1}^{n} CT_{ij}}{n} \tag{4.4}$$

The heuristic then finds the relative standard deviation[20].

$$SD' = \frac{SD}{avgCT} \tag{4.5}$$

The relative standard deviation shows the degree of dispersion of a set of values, here the set of values are CT_{ij}. If the value of the relative standard deviation is less than the critical value of relative standard deviation(ST), then task with minimum weighted mean time is chosen for mapping otherwise task with maximum weighted mean time is chosen for mapping. The critical value of relative standard deviation can be found by experiments, which come out to be 0.64 in this case.

4.3 Priority based QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

The priority based QoS guided weighted mean time min-min max-min selective heuristic (PQWMTS) is based on the weighted mean time min-min max-min selective heuristic[18]. The PQWMTS uses the priority grouping strategy to group the similar tasks. The heuristic creates n groups. These groups are created on the basis of services provided by resources at the time of mapping. Each group is assigned a priority value. The group having resource with highest QoS is assigned highest priority value. Each task is assigned to one of the groups based on its QoS demands or priority. The heuristic is more suitable in the environment where the resource heterogeneity is high and the task heterogeneity is also high. The heuristic is shown in figure (4.3). The working of PQWMTS is described in detail below.

```
(1)   Compute n QoS groups
(2)   Find the performance metric of each resource
(3)   While (i < n)
(4)      For each QoS group
(5)        Find the expected completion time of each task
(6)        Find the weighted mean time of each task
(7)        Compute the standard deviation SD
(8)        Compute relative standard deviation SD'
(9)        If SD' < ST then
(10)         Find task t_i having minimum weighted mean
             execution time and assign it to the resource,
             from the QoS qualified set, that is
             giving minimum completion time
           Else
(11)         Find task t_i having maximum weighted mean
             execution time and assign it to the resource,
             from the QoS qualified set, that is giving
             minimum completion time
(12)       Delete task t_i from the metatask
(13)       Update ready time of resource r_j
(14)     End For
(15)  End while
```

Figure 4.3 Priority based QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

i) **Compute n QoS groups:** - If there are n resources and each one is providing different QoS services, then n groups can be created. The grouping helps in satisfying users QoS demands. Each group is assigned a priority. The group having tasks with high utility value is assigned highest priority. Each task, based on its QoS demand, is assigned to one of the groups.

ii) **Find the performance metric of resource:** - The performance metric of each resource is computed as given in section 3.1 of chapter 3. Performance metric

is computed group wise. The metric shows the performance of resource in a group.

iii) **Scheduling Process:** - In the descending order of priority of groups, the tasks from groups are mapped. For each group, the heuristic performs the following steps. It computes the expected completion time of each task. The expected completion time can be computed using equation (4.2). Next, it finds the weighted mean time of each task. The weighted mean time of each task can be computed using equation (4.1). The heuristic computes the standard deviation of all unassigned tasks. The standard deviation can be computed using equation (4.3). The heuristic finds the value of relative standard deviation. The value of relative standard deviation can be computed using equation (4.5). The relative standard deviation shows the degree of dispersion of a set of values, here the set of values are CT_{ij}. If the value of the relative standard deviation is less than the critical value of relative standard deviation(ST), then task with minimum weighted mean time is chosen for mapping otherwise task with maximum weighted mean time is chosen for mapping. The critical value of relative standard deviation can be found by experiments, which come out to be 0.64 in this case.

4.4 Multiple QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

The multiple QoS guided weighted mean time min-min max-min selective (MQWMTS) heuristic considers multiple QoS demands such as deadline, response time, cost, priority etc. of tasks. The MQWMTS heuristic is shown in figure 4.4. The steps are described in detail below.

i) **Find the total utility of task:** - Let each task can request total d_i QoS. The equation given in (4.6) is used to find the total utility of a task.

$$U(t_i) = \left(\sum_{j=1}^{d_i} w_j u_j\right) \times \left({}^{p_i}/_{p_{max}}\right) \tag{4.6}$$

Here w_j represents the weight assigned to the utility u_j. The user can assign a weight to the utility and $\sum w_j=1$. User can give preferences to the various QoS needs by assigning different weight values to QoS parameters. P_i is the

priority of the task and P_{max} is the maximum priority assigned to the task. For the testing purpose response time[21], execution cost[21] and priority are chosen as QoS parameters. The cost parameter is modelled by following method. First, the cost value of each task on each resource is computed using equation (4.7).

$$c(i,j) = c_j \times ET(i,j) \tag{4.7}$$

Here c_j is the cost of execution per unit time. Then, the average execution cost of task t_i is computed using equation (4.8).

$$avgCost(t_i) = \frac{\sum_{j=1}^{m} c(i,j)}{m} \tag{4.8}$$

Here, m is the total resources which can satisfy the QoS requirements. Execution cost on each resource is computed and the minimum of it is chosen.

$$EC(i,j) = \frac{c(i,j)}{avgCost(t_i)} \tag{4.9}$$

For the response time QoS parameter, the following method is used. First, the response time of task t_i on every resource is computed using equation (4.10).

$$rt(i,j) = ft(i,j) - st(i,j) \tag{4.10}$$

Here, rt(i,j) is the response time of task t_i on resource r_j. ft(i,j) and st(i,j) is the finish and start time of task t_i on resource r_j, respectively. Next, the average execution time of task t_i is computed using equation (4.11).

$$avgET(t_i) = \frac{\sum_{j=1}^{m} rt(i,j)}{m} \tag{4.11}$$

Here, m is the resources which can satisfy the QoS requirements. The minimum value of avgET(t_i) is chosen. For the priority QoS parameter, the priority value is generated in the range from 1 to 4.

```
(1)   Find the total utility of each task.
(2)   Divide the tasks in n groups.
(3)   Find the performance metric of each resource
(4)   While (i < n)
(5)       For each QoS group
(6)         Find the expected completion time of each
            task
(7)         For each task t_i, compute the weighted mean
            time
(8)         Compute the standard deviation (SD)
(9)         Compute relative standard deviation SD
(10)        If SD' < ST then
(11)          Find task t_i having minimum weighted mean
              execution time and assign it to the
              resource, from the QoS qualified set, that
              is giving minimum completion time
            Else
(12)          Find task t_i having maximum weighted
              mean execution time and assign it to the
              resource, from the QoS qualified set, that
              is giving minimum completion time
(13)      Delete task t_i from the metatask
(14)      Update ready time of resource r_j
(15)  End while
```

Figure 4.4 Multiple QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

ii) **Divide the task in n groups:** - The tasks are divided in number of groups on the basis of their total utility value. The division helps to give priority to high QoS demanding tasks.

iii) **Find the performance metric of each resource:** - The performance metric of each resource is computed as given in section 3.1 of chapter 3. Performance

metric is computed group wise. The metric shows the performance of resource in a group.

iv) **Scheduling Process: -** For each group the heuristic performs the following steps. It calculates the weight of the resources in that group. It computes the expected completion time of each task on each resource. It computes the standard deviation of the completion time of unassigned tasks of metatask. The standard deviation can be found using equation(4.3). Which task, having maximum or minimum weighted mean time, will be chosen for the mapping that depends on the critical value of the relative standard deviation(SD'). The relative standard deviation can be found using equation (4.5). The relative standard deviation shows the degree of dispersion of a set of values, here the set of values are CT_{ij}. If the value of the relative standard deviation is less than the critical value of relative standard deviation(ST), then task with minimum weighted mean time is chosen for mapping otherwise task with maximum weighted mean time is chosen for mapping. The critical value of relative standard deviation can be found by experiments, which come out to be 0.64 in this case.

Chapter 5

Simulation Tool and Performance Metrics

5.1 GridSim: Grid Modeling and Simulation Toolkit

The GridSim[22] toolkit provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers. It can be used to simulate application schedulers for single or multiple administrative domains distributed computing systems such as clusters and Grids. Application schedulers in the Grid environment, called resource brokers, perform resource discovery, selection, and aggregation of a diverse set of distributed resources for an individual user. This means that each user has his or her own private resource broker and hence it can be targeted to optimize for the requirements and objectives of its owner. In contrast, schedulers, managing resources such as clusters in a single administrative domain, have complete control over the policy used for allocation of resources. This means that all users need to submit their jobs to the central scheduler, which can be targeted to perform global optimization such as higher system utilization and overall user satisfaction depending on resource allocation policy or optimize for high priority users. GridSim is better for simulating the grid based algorithms because

- It allows modeling of heterogeneous types of resources.
- Resources can be modeled in two modes : space shared and time shared.
- Resource capability can be defined in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark.
- Advance reservation of resources can be done.
- Application tasks can be heterogeneous and they can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared.
- Network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.

- Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

5.2 GridSim Entities

GridSim support various entities for simulating the heterogeneous resources that can be configured as time or space shared systems. The various entities are given below

i) **User:** - Each user represents a grid user. Each user can create different jobs. Each user can define its scheduling optimization policy either minimizing time or cost or both.

ii) **Broker:** - Broker is the entity which schedules the jobs on resources. The broker gathers information about available resources and schedules the jobs submitted by users. Broker tries to optimize the scheduling policy.

iii) **Resource:** - The grid resources differ from each other on following characteristics:

   a. Number of processing elements
   b. Cost of processing
   c. Speed
   d. Internal scheduling policy : time or space shared

The resource speed and the job execution time can be defined in terms of the ratings of standard benchmarks such as MIPS and SPEC.

iv) **Grid Information Service:** - Grid information service is responsible for providing resource registration and keeping track of available resources in grid. The broker can query grid information service for resource's configuration and status information.

v) **Input and Output:** - The information from one entity to another entity flows though its input and output entities. The I/O channels or ports are used to establish link between entity and its input and output entities. The support for buffered input and output channels associated with every GridSim entity provides a simple mechanism for an entity to communicate with other entities and at the same time enables modeling of the necessary communications delay transparently.

5.3 Simulation Environment and Data

The GridSim5.0 simulation toolkit is used to simulate and validate the proposed heuristics. The proposed as well as QoS guided min-min, weighted mean time-min, weighted mean time min-min max-min selective, min-min and max-min heuristics are implemented in gridsim. For each experiment, the batch size of tasks is taken as 1000 tasks. 20 resources are taken for each experiment. Each resource is having a number of machines. This number of machines is generated randomly in the range from 5 to 10. Each machine is having a number of processing elements. This number of processing elements is generated randomly in the range from 5 to 10. The task arrival is modelled as poison random process. For each experiment, the following task scenarios are taken.

Scenario I: A few short tasks with many long tasks.

Scenario II: A few long tasks with many short tasks.

Scenario III: The task length is determined randomly.

5.4 Performance Metrics

Depending on what scheduling performance is desired in grid there exists various performance metrics for evaluating scheduling algorithms. The results of the proposed heuristics are evaluated on the basis of following performance matrices:

5.4.1 Makespan: - Makespan is a measure of throughput of the Grid. It can be calculated using equation

$$\text{makespan} = \max_{t_i \in MT}(CT_i) \tag{5.1}$$

Here, CT_i is the completion time of task t_i. The less the makespan the better the algorithm is.

5.4.2 Load balance degree: - The load balance degree[23], β, is determined through the relative deviation of mean square deviation, d, of resource utilization.

$$\beta = 1 - \frac{d}{ru} \tag{5.2}$$

The best load balancing level is achieved if β reaches to 1 and d is close to 0. The mean square deviation, d, of average resource utilization(ru) is given by equation (5.3)

$$d = \sqrt{\frac{\sum_{i=1}^{m}(ru - ru_j)^2}{m}} \tag{5.3}$$

Here ru is the average resource utilization resources, which can be computed using equation (5.4)

$$ru = \frac{\sum_{j=1}^{m} ru_j}{m} \tag{5.4}$$

Here ru_j is the resource utilization of resource r_j and can be computed using equation (5.5)

$$ru_j = \frac{\sum_{i \text{ where } t_i \text{has been executed on } m_j}(te_i - ts_i)}{T} \tag{5.5}$$

Here te_i is the finish time and ts_i is the start time of task t_i on resource r_j. T is the total application time elapsed so far and can be computed using equation (5.6)

$$T = \max(te_i) - \min(ts_i) \tag{5.6}$$

The load balance degree shows the resource load balancing achieved by a heuristic. Both, makespan and load balance degree, are used to validate the proposed heuristics.

Chapter 6

Results and Discussions

The task scenarios given in section 5.3 of chapter 5 are used to test the all proposed heuristics. The simulation environment as given in section 5.3 of chapter 5 is created for every heuristics. The results of makespan and load balance degree of every heuristic are computed. The obtained results of proposed heuristics are compared with the results of other heuristics such as QoS guided min-min, weighted mean time-min, weighted mean time min-min max-min selective, min-min and max-min. The detailed results are shown below.

6.1 Results of Segmented Weighted Time-Min Heuristic

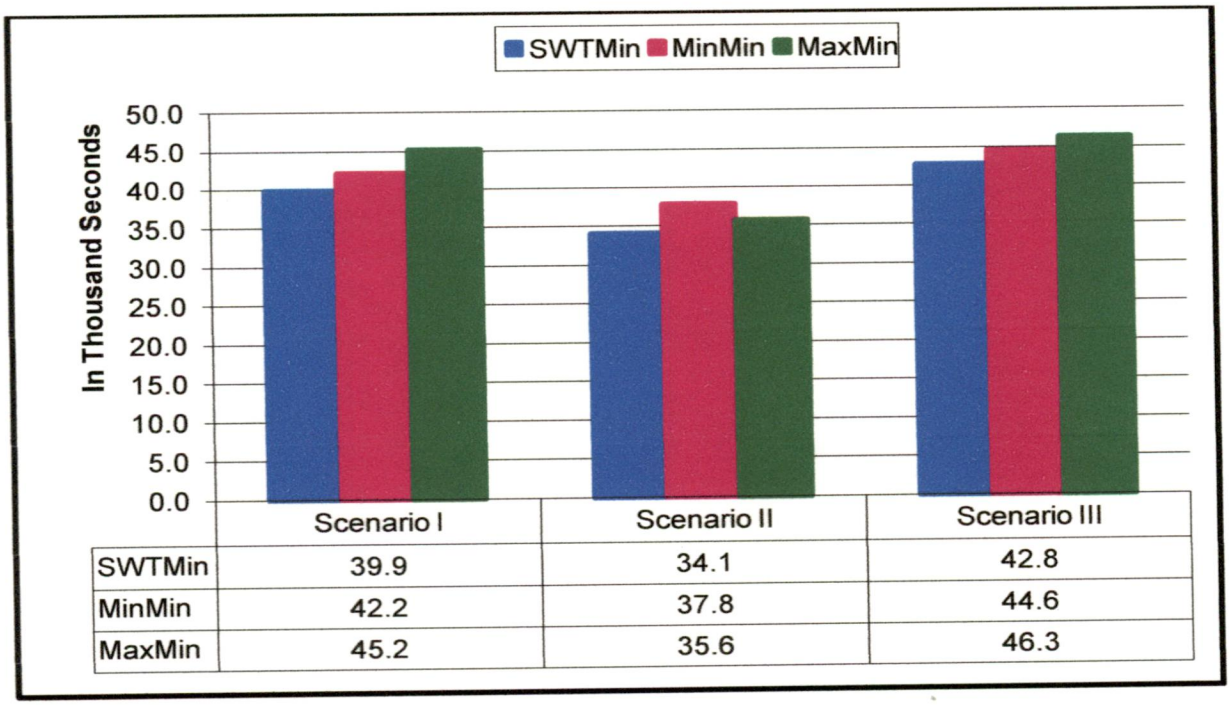
a) **Makespan:** - The makespan results are shown in figure 6.1. The makespan results are compared with the results of Min-Min and Max-Min heuristics. Table 6.1 shows the comparison of Segmented Weighted Time-Min, Min-Min and Max-Min Heuristics. The SWT-Min heuristics gives 5.45%, 9.78% and 4.03% improvement in makespan than Min-Min heuristic for the task scenario I, II and III, respectively. The SWT-Min heuristics gives 11.72%, 4.21% and 7.55% improvements in makespan than Max-Min heuristic for the task scenario I, II and III, respectively. Overall, SWT-Min heuristic gives better makespan than Min-Min and Max-Min heuristic for each task scenario.

Table 6.1 Makespan Comparison of SWT-Min, Min-Min and Max-Min Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement over MinMin	Improvement over MaxMin
	SWT-Min	MinMin	MaxMin		
I	39.9	42.2	45.2	5.45%	11.72%
II	34.1	37.8	35.6	9.78%	4.21%
III	42.8	44.6	46.3	4.03%	7.55%

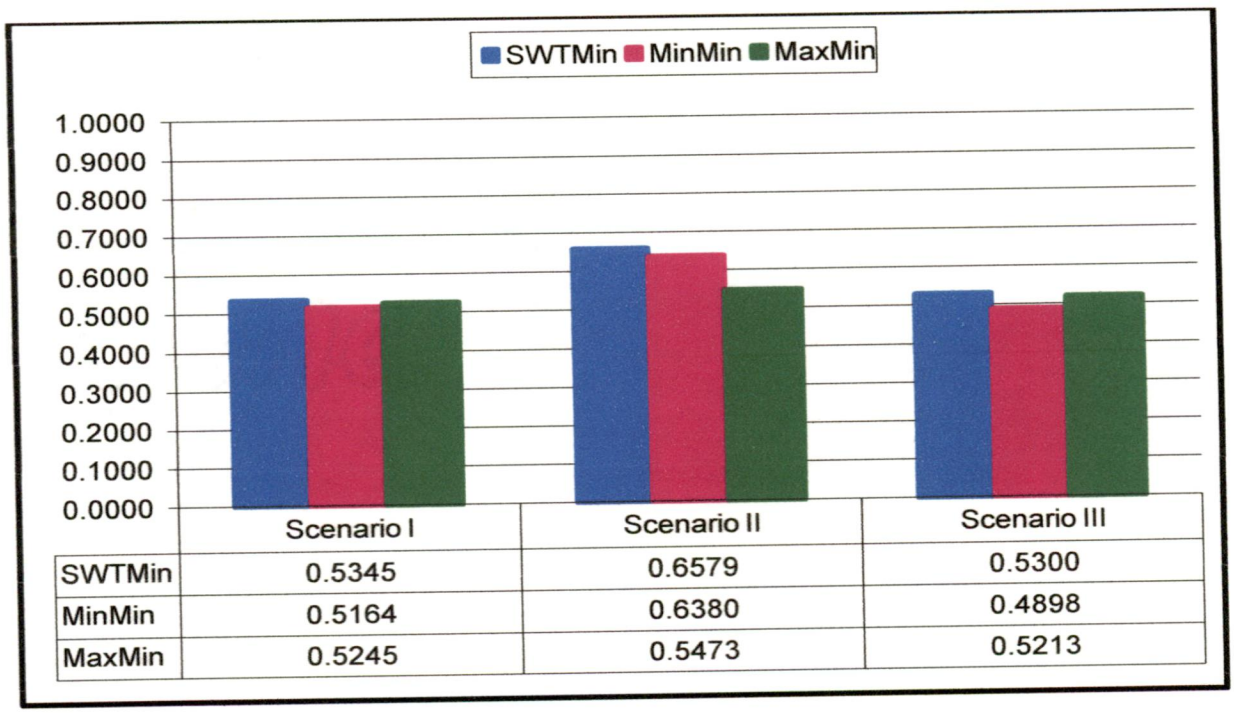
b) **Load Balance Degree:** - The load balance degree of SWT-Min is shown in figure 6.2. The results are compared with the results of Min-Min and Max-Min heuristics. The load balance degree shows resource load balance across the resources. We can see from the figure that the proposed heuristic, SWT-

Min, has achieved maximum load balance in each task scenario I, II and III than Min-Min and Max-Min heuristics. We can conclude that the proposed heuristic is better in load balancing than Min-Min and Max-Min heuristics.



	Scenario I	Scenario II	Scenario III
SWTMin	39.9	34.1	42.8
MinMin	42.2	37.8	44.6
MaxMin	45.2	35.6	46.3

Figure 6.1 Makespan of SWT-Min, Min-Min and Max-Min Heuristics

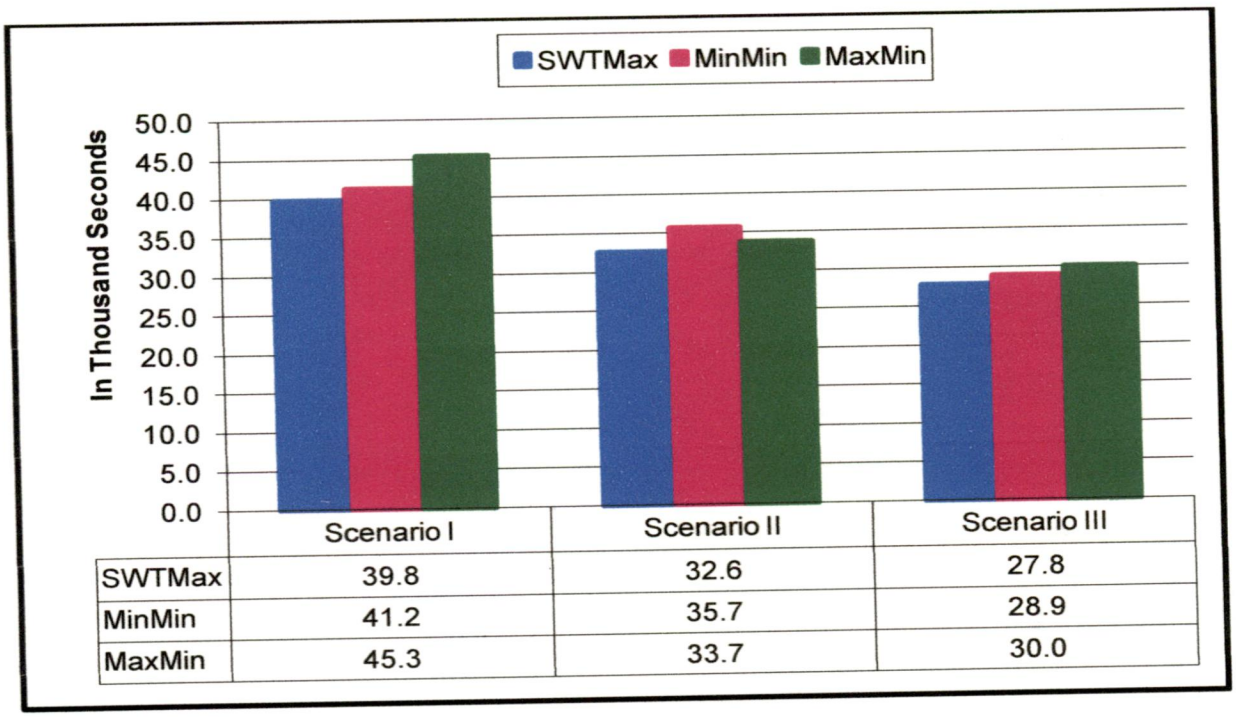


	Scenario I	Scenario II	Scenario III
SWTMin	0.5345	0.6579	0.5300
MinMin	0.5164	0.6380	0.4898
MaxMin	0.5245	0.5473	0.5213

Figure 6.2 Load Balance Degree of SWT-Min, Min-Min and Max-Min Heuristics

6.2 Results of Segmented Weighted Time-Max Heuristic

a) Makespan: - The makespan results are shown in figure 6.3. The makespan results are compared with the results of Min-Min and Max-Min heuristics. Table 6.2 shows the comparison of Segmented Weighted Time-Max, Min-Min and Max-Min Heuristics. The SWT-Max heuristics gives 3.39%, 8.68% and 3.8% less makespan than Min-Min heuristic for the task scenario I, II and III, respectively. The SWT-Max heuristics gives 12.14%, 3.26% and 7.33% less makespan than Max-Min heuristic for the task scenario I, II and III, respectively. Overall the proposed heuristic SWT-Max gives better makespan than Min-Min and Max-Min heuristic for each task scenario.



	Scenario I	Scenario II	Scenario III
SWTMax	39.8	32.6	27.8
MinMin	41.2	35.7	28.9
MaxMin	45.3	33.7	30.0

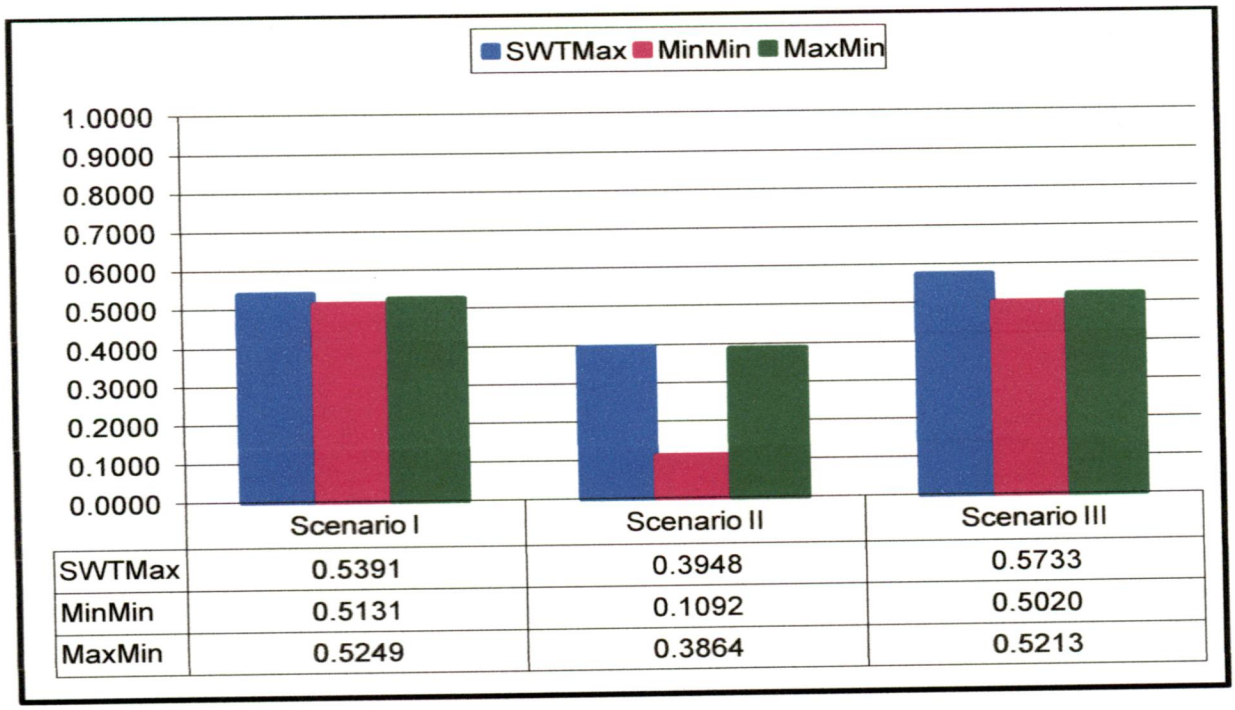
Figure 6.3 Makespan of SWT-Max, Min-Min and Max-Min Heuristics

Table 6.2 Makespan Comparison of SWT-Max, Min-Min and Max-Min Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement Over MinMin	Improvement Over MaxMin
	SWT-Max	MinMin	MaxMin		
I	39.8	41.2	45.3	3.39%	12.14%
II	32.6	35.7	33.7	8.68%	3.26%
III	27.8	28.9	30.0	3.8%	7.33%

b) Load Balance Degree: - The load balance degree results of SWT-Max are shown in figure 6.4. The results are compared with the results of Min-Min

and Max-Min heuristics. We can see from the figure that the proposed heuristic, SWT-Max, has achieved maximum load balance in each task scenario I, II and III than Min-Min and Max-Min heuristics. We can conclude that the proposed heuristic is better in load balancing than Min-Min and Max-Min heuristics.



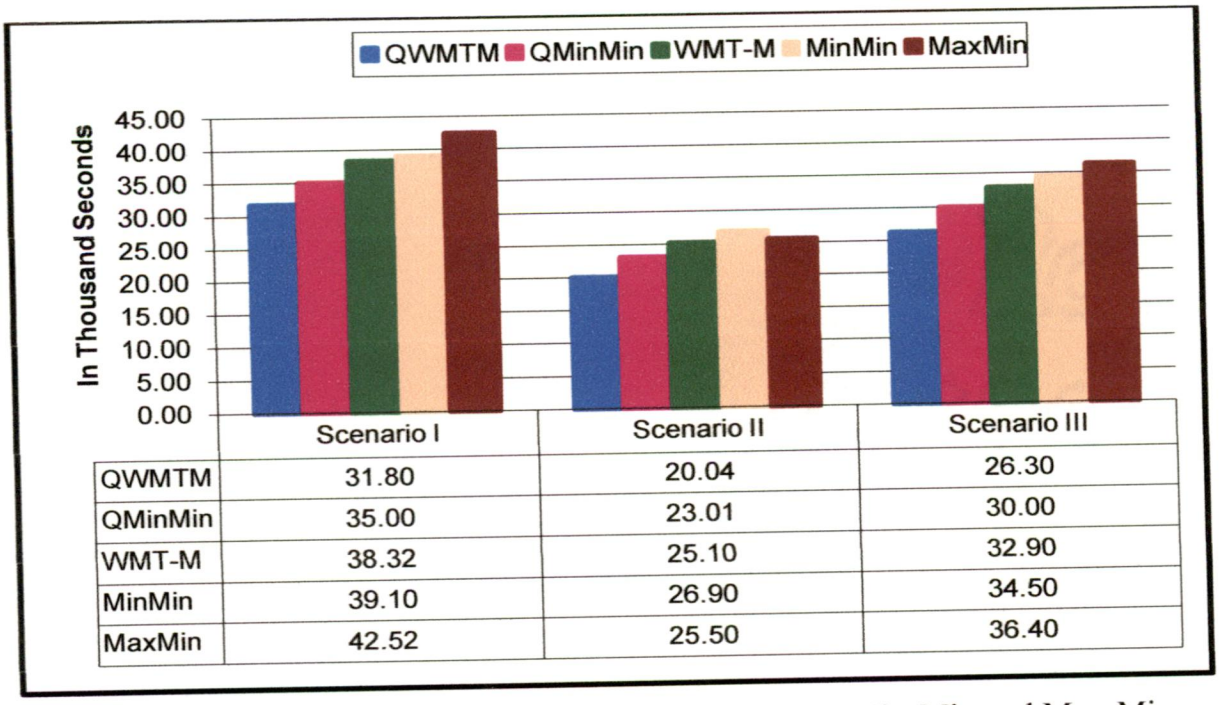
	Scenario I	Scenario II	Scenario III
SWTMax	0.5391	0.3948	0.5733
MinMin	0.5131	0.1092	0.5020
MaxMin	0.5249	0.3864	0.5213

Figure 6.4 Load Balance Degree of SWT-Max, Min-Min and Max-Min Heuristics

6.3 Results of QoS Guided Weighted Mean Time-Min Heuristic

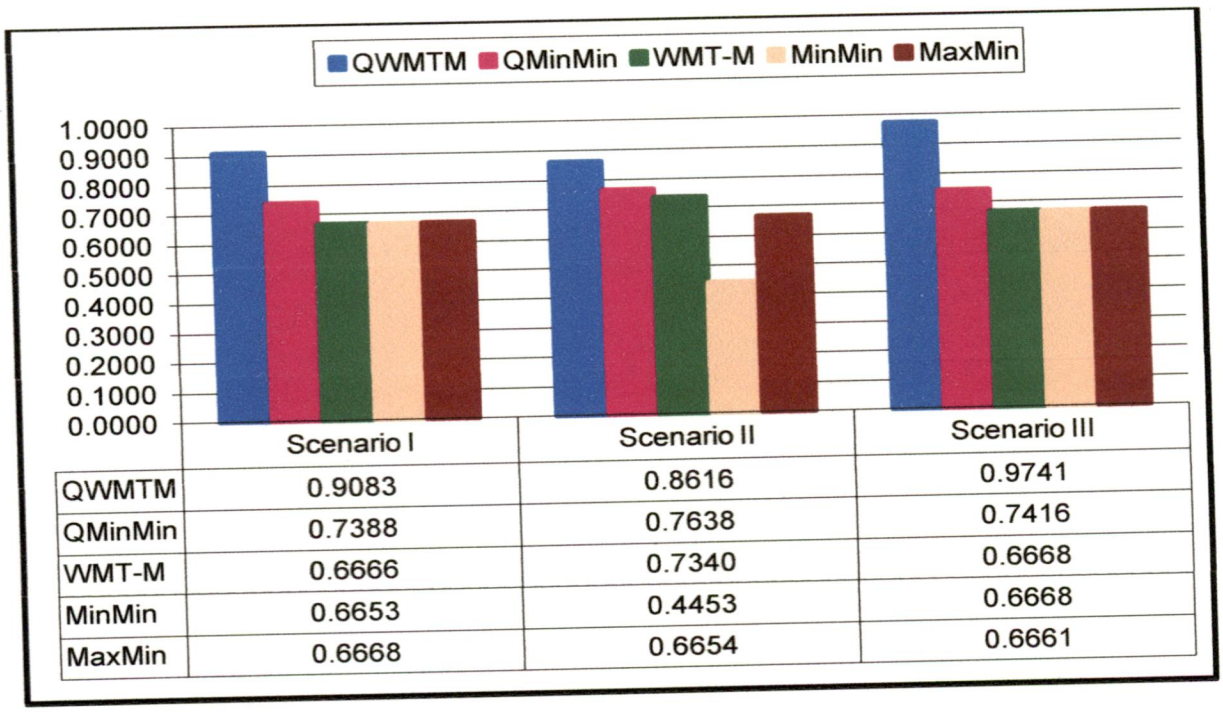
a) Makespan: - The makespan results of QoS Guided Weighted Mean Time-Min (QWMTM), QoS Guided Min-Min(QMinMin), Weighted Mean Time-Min(WMT-M), Min-Min and Max-Min for the task scenario I, II and III are shown in figure 6.5. The comparison of makespan results is shown in table 6.3 and 6.4. The QWMTM heuristic gives 9.14%, 12.9%, 12.33% improvements in makespan than QMinMin heuristic for the task scenario I, II and III, respectively. It gives 17%, 20.16%, 20% improvements in makespan than WMT-M for the task scenario I, II and III, respectively. It gives 18.67%, 25.5%, 23.77% improvements in makespan than Min-Min heuristic for the task scenario I, II and III, respectively. It gives 25.21%, 21.41%, 27.74% improvements in makespan than Max-Min for the task scenario I, II and III,

respectively. We can conclude that QWMTM outperforms in each task scenario than QMinMin, WMT-M, Min-Min and Max-Min heuristics.



	Scenario I	Scenario II	Scenario III
QWMTM	31.80	20.04	26.30
QMinMin	35.00	23.01	30.00
WMT-M	38.32	25.10	32.90
MinMin	39.10	26.90	34.50
MaxMin	42.52	25.50	36.40

Figure 6.5 Makespan of QWMTM, QMinMin, WMT-M, Min-Min and Max-Min Heuristics



	Scenario I	Scenario II	Scenario III
QWMTM	0.9083	0.8616	0.9741
QMinMin	0.7388	0.7638	0.7416
WMT-M	0.6666	0.7340	0.6668
MinMin	0.6653	0.4453	0.6668
MaxMin	0.6668	0.6654	0.6661

Figure 6.6 Load Balance Degree of QWMTM, QMinMin, WMT-M, Min-Min and Max-Min Heuristics

b) **Load Balance Degree:** - The load balance degree of QWMTM, QMinMin, WMT-M, Min-Min and Max-Min are shown in figure 6.6. We can see from the figure that the QWMTM heuristic performs better load balancing than WMT-M, QMinMin, Min-Min and Max-Min heuristics.

Table 6.3 Makespan Comparison of QWMTM, QMinMin and WMT-Min Heuristics

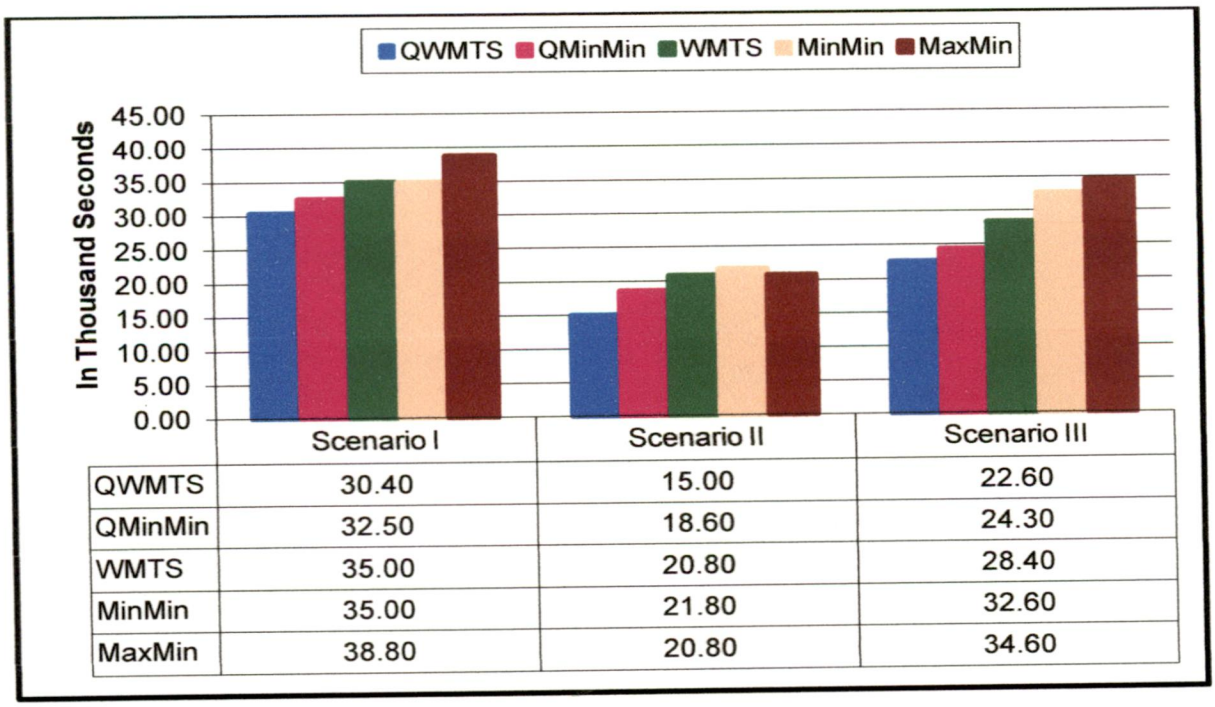
Task Scenario	Makespan (In Thousand Seconds)			Improvement Over QMinMin	Improvement Over WMT-M
	QWMTM	QMinMin	WMT-M		
I	31.8	35	38.32	9.14%	17%
II	20.04	23.01	25.1	12.9%	20.16%
III	26.30	30	32.9	12.33%	20%

Table 6.4 Makespan Comparison of QWMTM, Min-Min and Max-Min Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement Over MinMin	Improvement Over Max-Min
	QWMTM	Min-Min	Max-Min		
I	31.8	39.1	42.52	18.67%	25.21%
II	20.04	26.9	25.5	25.5%	21.41%
III	26.30	34.5	36.4	23.77%	27.74%

6.4 Results of QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic

a) **Makespan:** - The makespan results of QoS Guided Weighted Mean Time Min-Min Max-Min Selective (QWMTS), QMinMin, Weighted Mean Time Min-Min Max-Min Selective (WMTS), Min-Min and Max-Min heuristics are shown in figure 6.7. Table 6.5 and 6.6 show the comparison of makespan results. The QWMTS gives 6.46%, 19.35%, 6.99% gain in makespan than QMinMin for the task scenario I, II and III, respectively. It gives 13.14%, 27.88%, 20.42% gain in makespan than WMTS for the task scenario I, II, and III, respectively. It gives 13.14%, 31.19%, 30.67% gain in makespan than Min-Min for the task scenario I, II, and III, respectively. It gives 21.65%, 27.88%, 34.68% gain in makespan than Max-Min for the task scenario I, II, and III, respectively. We can conclude that the proposed heuristic gives better makespan than other heuristics for each task scenario.



	Scenario I	Scenario II	Scenario III
QWMTS	30.40	15.00	22.60
QMinMin	32.50	18.60	24.30
WMTS	35.00	20.80	28.40
MinMin	35.00	21.80	32.60
MaxMin	38.80	20.80	34.60

Fig 6.7 Makespan of QWMTS, QMinMin, WMTS, Min-Min and Max-Min Heuristics

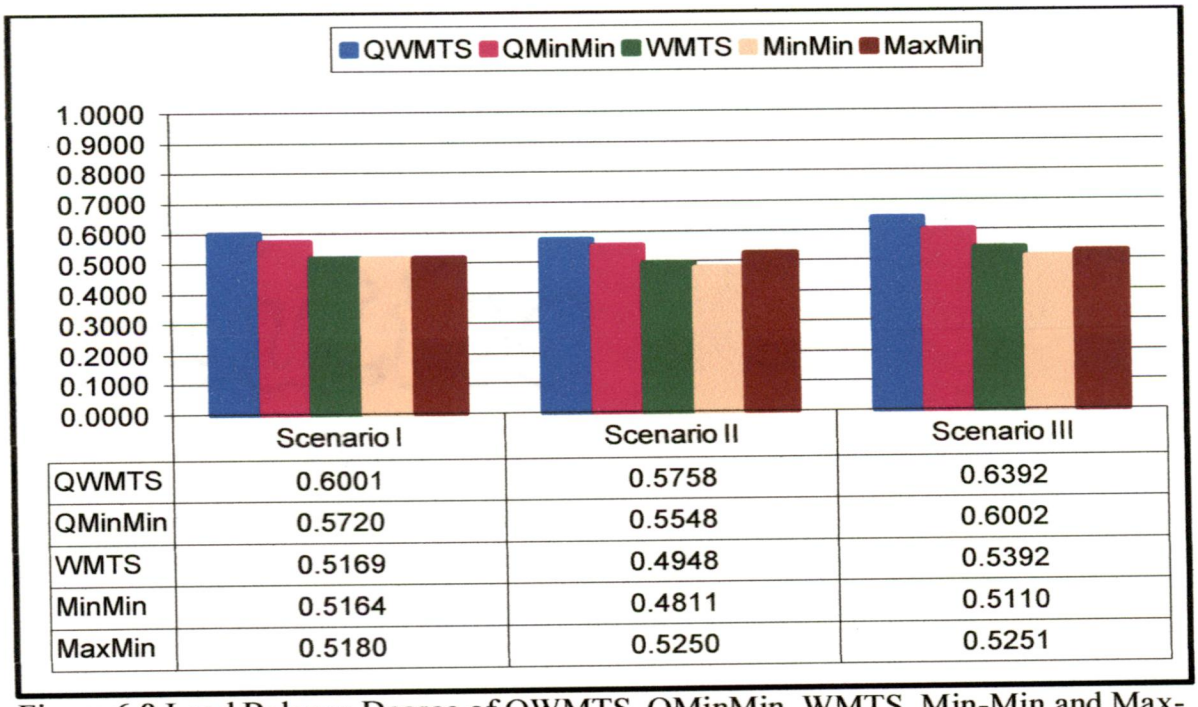
Table 6.5 Makespan Comparison of QWMTS, QMinMin and WMTS Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement Over QMinMin	Improvement Over WMTS
	QWMTS	QMin-Min	WMTS		
I	30.4	32.5	35	6.46%	13.14%
II	15	18.6	20.8	19.35%	27.88%
III	22.6	24.3	28.4	6.99%	20.42%

Table 6.6 Makespan Comparison of QWMTS, Min-Min and Max-Min Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement Over MinMin	Improvement Over MaxMin
	QWMTS	Min-Min	Max-Min		
I	30.4	35	38.8	13.14%	21.65%
II	15	21.8	20.8	31.19%	27.88%
III	22.6	32.6	34.6	30.67%	34.68%

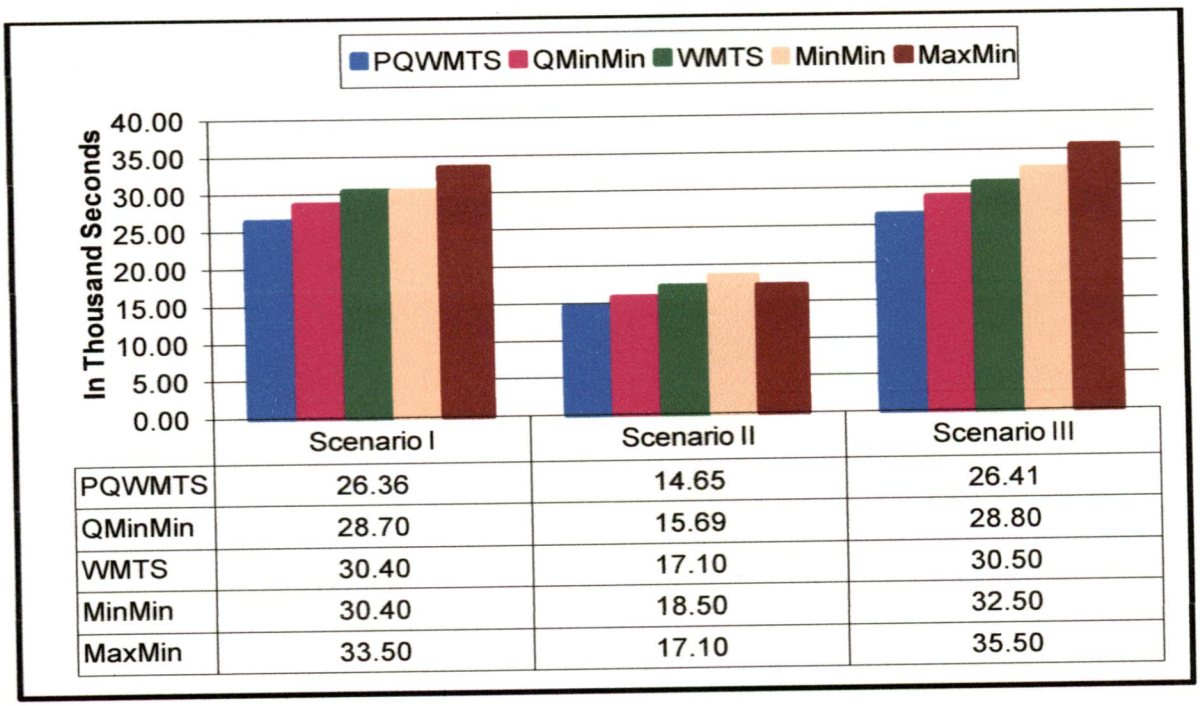
b) **Load Balance Degree:** - The load balance degree of QWMTS, QMinMin, WMTS, Min-Min and Max-Min heuristics are given in figure 6.8. We can see from the figure that the proposed heuristic QWMTS gives better load balancing than QMinMin, WMTS, Min-Min and Max-Min heuristics. The QWMTS achieves better load balancing in each task scenario.



	Scenario I	Scenario II	Scenario III
QWMTS	0.6001	0.5758	0.6392
QMinMin	0.5720	0.5548	0.6002
WMTS	0.5169	0.4948	0.5392
MinMin	0.5164	0.4811	0.5110
MaxMin	0.5180	0.5250	0.5251

Figure 6.8 Load Balance Degree of QWMTS, QMinMin, WMTS, Min-Min and Max-Min Heuristics

6.5 Results of Priority based QoS Guided Weighted Mean Time Min-Min Max-Min Selective Heuristic



	Scenario I	Scenario II	Scenario III
PQWMTS	26.36	14.65	26.41
QMinMin	28.70	15.69	28.80
WMTS	30.40	17.10	30.50
MinMin	30.40	18.50	32.50
MaxMin	33.50	17.10	35.50

Figure 6.9 Makespan of PQWMTS, QMinMin, WMTS, Min-Min and Max-Min Heuristics

a) **Makespan:** - The makespan results of priority based PQWMTS, QMinMin, WMTS, Min-Min and Max-Min heuristics are shown in figure 6.9. Table 6.7 and 6.8 are showing the comparison of makespan results. The PQWMTS gives 8.15%, 6.62%, 8.29% gain in makespan than QMinMin for the task scenario I, II, and III, respectively. It gives 13.29%, 14.33%, 13.41% gain in makespan than WMTS for the task scenario I, II and III, respectively. It gives 13.29%, 20.81%, 18.74% gain in makespan than Min-Min for the task scenario I, II and III, respectively. It gives 21.31%, 14.33%, 25.6% gain in makespan than Max-Min for the task scenario I, II and III, respectively. We can conclude that the proposed heuristic gives better makespan for each task scenario than other heuristics.

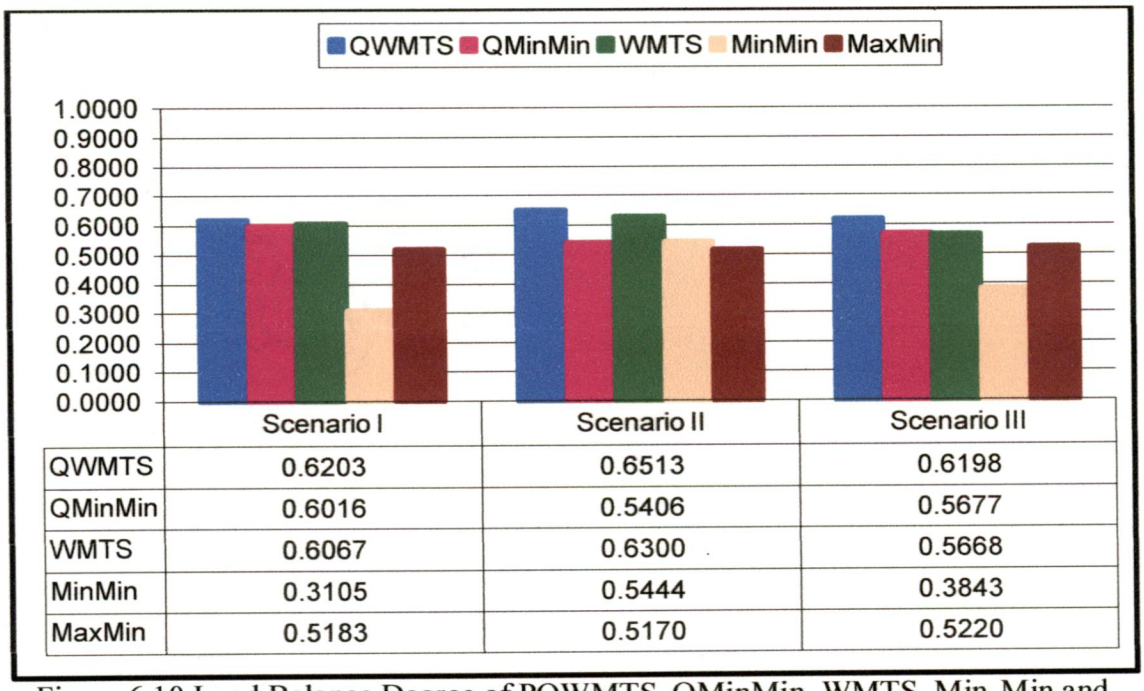
b) **Load Balance Degree:** - The load balance degree of PQWMTS, QMinMin, WMTS, Min-Min and Max-Min are shown in figure 6.10. The PQWMTS achieves best load balance in each task scenario.

Table 6.7 Makespan Comparison of PQWMTS, QMinMin and WMTS Heuristics

Task Scenario	Makespan (In Thousand Seconds)			Improvement Over QMinMin	Improvement Over WMTS
	PQWMTS	QMinMin	WMTS		
I	26.36	28.7	30.4	8.15%	13.29%
II	14.65	15.69	17.01	6.62%	14.33%
III	26.41	28.8	30.5	8.29%	13.41%

Table 6.8 Makespan Comparison of PQWMTS, Min-Min and Max-Min Heuristics

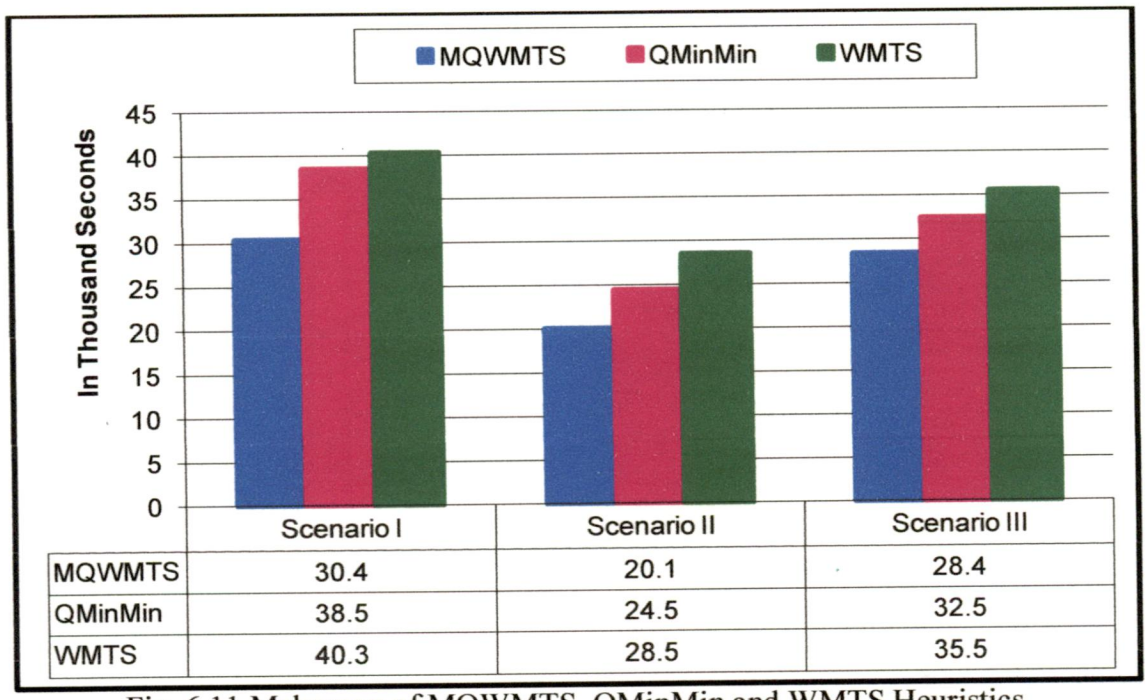
Task Scenario	Makespan (In Thousand Seconds)			Improvement Over MinMin	Improvement Over MaxMin
	PQWMTS	Min-Min	Max-Min		
I	26.36	30.4	33.5	13.29%	21.31%
II	14.65	18.5	17.1	20.81%	14.33%
III	26.41	32.5	35.5	18.74%	25.6%



	Scenario I	Scenario II	Scenario III
QWMTS	0.6203	0.6513	0.6198
QMinMin	0.6016	0.5406	0.5677
WMTS	0.6067	0.6300	0.5668
MinMin	0.3105	0.5444	0.3843
MaxMin	0.5183	0.5170	0.5220

Figure 6.10 Load Balance Degree of PQWMTS, QMinMin, WMTS, Min-Min and Max-Min Heuristics

6.6 Results of Multiple QoS Guided Weighted Mean Time Min-Min Max-Min Heuristic



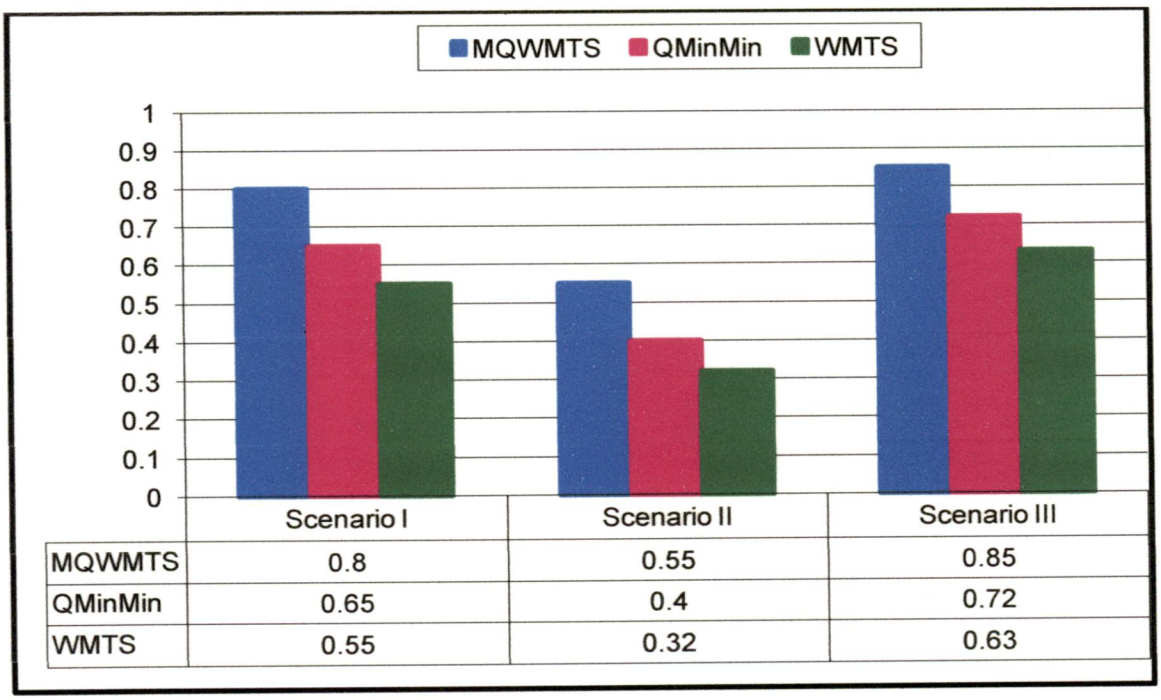
	Scenario I	Scenario II	Scenario III
MQWMTS	30.4	20.1	28.4
QMinMin	38.5	24.5	32.5
WMTS	40.3	28.5	35.5

Fig. 6.11 Makespan of MQWMTS, QMinMin and WMTS Heuristics

a) **Makespan:** - Figure 6.11 shows the makespan results of MQWMTS, QMinMin and WMTS heuristics. Table 6.9 shows the comparison of

makespan results. We can see from the table that the MQWMTS gives significant improvements than QMinMin and WMTS. MQWMTS gives 21%, 17.95%, 12% improvement in makespan than QMinMin for the task scenario I, II and III, respectively. It gives 24.56%, 29%, 20% gain in makespan than WMTS for task scenario I, II and III, respectively. Overall, the MQWMTS heuristic outperforms in each task scenario than QMinMin and WMTS heuristics.

b) **Load Balance Degree:** - Figure 6.12 shows the load balance degree of MQWMTS, QMinMin and WMTS heuristics for the task scenario I, II, and III, respectively. We can see from the figure that MQWMTS gives better load balancing in each task scenario.



	Scenario I	Scenario II	Scenario III
MQWMTS	0.8	0.55	0.85
QMinMin	0.65	0.4	0.72
WMTS	0.55	0.32	0.63

Fig. 6.12 Load Balance Degree of MQWMTS, QMinMin and WMTS Heuristics

Table 6.9 Makespan Comparison of MQWMTS, QMinMin and WMTS

Task Scenario	Makespan (In Thousand Seconds)			Improvement over QMinMin	Improvement over WMTS
	MQWMTS	QMinMin	WMTS		
I	30.4	38.5	40.3	21%	24.56%
II	20.1	24.5	28.5	17.95%	29%
III	28.4	32.5	35.5	12%	20%

Chapter 7

Conclusions and Scope for Future Work

7.1 Conclusions

In this dissertation work, two resource performance based and four QoS based heuristics are proposed. The segmented weighted time-min and segmented weighted time-max heuristics are based on resource performance. The QoS guided weighted mean time-min, QoS guided weighted mean time min-min max-min selective, priority based QoS guided weighted mean time min-min max-min selective and multiple QoS guided weighted mean time min-min max-min selective scheduling heuristics are based on QoS. All heuristics are tested on the basis of makespan and load balance degree performance metrics. The following improvements in makespan are obtained.

- The segmented weighted time-min gives up to 9.78% and 11.72% improvement in makespan than min-min and max-min heuristics, respectively.
- The segmented weighted time-max gives up to 8.68% and 12.14% improvement in makespan than min-min and max-min heuristics, respectively.
- The QWMTM heuristic gives up to 12.9%, 20.16%, 25.5% and 27.74% improvement in makespan than QoS guided min-min, WMT-M, min-min and max-min heuristics, respectively.
- The QWMTS heuristic gives up to 19.35%, 27.88%, 31.19% and 34.68% improvement in makespan than QoS guided min-min, WMTS, min-min and max-min heuristics, respectively.
- The PQWMTS heuristic gives up to 8.29%, 14.33%, 20.81% and 25.6% improvement in makespan than QoS guided min-min, WMTS, min-min and max-min heuristics, respectively.
- The MQWMTS heuristic gives up to 17.95% and 29% improvement in makespan than QoS guided min-min and WMTS heuristics, respectively.

All the heuristics are also tested for resource load balancing. From the results given in chapter 6, we can conclude that all proposed heuristics do better load balancing across the resources.

7.2 Scope for Future Work

In this dissertation work, two resource performance based and four QoS based scheduling heuristics are proposed. The proposed heuristics are tested for independent tasks batch mode scheduling in static environment. The following domains can be considered for future work.

i) The heuristics can be investigated in dynamic environment.

ii) The heuristics can be implemented and tested in actual grid environment.

iii) More QoS parameters like availability, trust, etc. can be considered to test the MQWMTS heuristic.

REFERENCES

[1] Ian Foster, Carl Kesselman, "THE GRID 2: BLUEPRINT FOR A NEW COMPUTING INFRASTRUCTURE", Second Edition, Margan Kaufmann, Published, 2004.

[2] D. Fernández-Baca, "Allocating Modules to Processors in a Distributed System," IEEE Transactions on Software Engineering, pp. 1427-1436, November 1989

[3] Maheswaran M, Ali S, Siegel H J, et al, "Dynamic Mapping of a Class of Independent tasks onto Heterogeneous Computing Systems", 8th IEEE Heterogeneous Computing Workshop (HCW '99), pp.30-44, Apr. 1999.

[4] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," Procceding of the 9th heterogeneous Computing Workshop (HCW'00), pp. 349-363, May 2000.

[5] Xiao-Shan He, Xian-He Sun, "QoS Guided Min-Min Heuristic for Grid Task Scheduling", Jouranal of Computer Science & Technology, 2003, (5): 442-451, 2003

[6] Fang Dong, Junzhou Luo, Lisha Gao and Liang Ge, "A Grid Task Scheduling Algorithm Based on QoS Priority Grouping", Fifth International Conference of Grid and Cooperative Computing, pages 58-61,2006.

[7] M. Baker, R. Buyya and D. Laforenza, "Grids and Grid Technologies for Wide-area Distributed Computing", Journal of Software-Practice & Experience, Vol. 32, No.15, , pp:1437-1466, December 2002.

[8] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing", Software: Practice and Experience (SPE) Journal, ISSN: 0038-0644, Volume 32, Issue 2, 2002, Wiley Press, USA, February 2002.

[9] F. Dong, S.G. Akl, Scheduling algorithms for grid computing: state of the art and open problems, Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, January 2006.

[10] J. M. Schopf, "A General Architecture for Scheduling in the Grid", Journal of parallel and distributed computing, special issue of Grid Computing, 2002

[11] T.D. Braun, H.J. Siegel, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, in: Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, pp. 330–335, October 1998.

[12] Barun TD, Siegel H J and Beck N. "A comparison of Eleven static heuristics for mapping a class of independent tasks onto Heterogeneous Distributed computing systems" Journal of Parallel and Distributed Computing Vol. 61, No. 1. pp. 810-837, 2001.

[13] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence, "Taxonomy for QoS Specifications. In Proceedings of WORDS, Newport Beach, CA, pp. 100–107, Feb. 1997.

[14] Golconda, K.S.; Ozguner, F.; Dogan, A., "A comparison of static QoS-based scheduling heuristics for a meta-task with multiple QoS dimensions in heterogeneous computing," Proceedings, 18th International Parallel and Distributed Processing Symposium, 2004, pp. 106, 26-30 April 2004.

[15] Xiaozhi Wang, Junshou Luo, "Architecture of Grid Allocation Management Based on QoS", Grid and Cooperative Computing, Springer LNCS, vol. 3033, pp. 81-88, 2004.

[16] Zhang Jinquan, Lina, N., Jiang Changjun, "A heuristic scheduling strategy for independent tasks on grid," Proceedings Eighth International Conference on High-Performance Computing in Asia-Pacific Region, 2005, pp.6 pp.-593, July 2005.

[17] M. Wu, W. Shu and H. Zhang, "Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems," Procceding of the 9th Heterogeneous Computing Workshop (HCW'00), pp. 375-385, Cancun, Mexico, May 2000.

[18] Sameer Singh Chauhan and R. C. Joshi, "A Weighted Mean Time Min-Min Max-Min Selective Scheduling Strategy for Independent Tasks on Grid", In proceedings of IEEE 2nd International Advance Computing Conference – 2010 (IACC 2010), pp. 4-9, 19-20, February, 2010.

[19] Etminani, K.; Naghibzadeh, M. , "A Min-Min Max-Min selective algorithm for grid task scheduling,", 3rd IEEE/IFIP International Conference in Central Asia on Internet 2007(ICI 2007), pp.1-7, 26-28 Sept. 2007.

[20] Yong Hou; Jiong Yu; Turgun; , "NDA-MM: A New Adaptive Task Scheduling Algorithm Based on the Non-dedicated Constraint Grid," Sixth International Conference on Grid and Cooperative Computing, 2007 (GCC 2007), pp.275-282, 16-18 Aug. 2007

[21] Saurabh Garg, Rajkumar Buyya and Howard J. Siegel, "Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management", Proceedings of the 32nd Australasian Computer Science Conference (ACSC 2009), ISBN 978-1-920682-72-9, Australian Computer Society, pp 139-147, January 19-23, 2009.

[22] R. Buyya, M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," Journal of Concurrency and Computation: Practice and Experience, pp. 1175–1220, 2002.

[23] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, "Grid Load Balancing Using Intelligent Agents," Future Generation Computer Systems, vol. 21 , Issue 1 ,January 2005.

LIST OF PUBLICATIONS

[1] Sameer Singh Chauhan and R. C. Joshi, ***"QoS Guided Heuristic Algorithms for Grid Task Scheduling"***, International Journal of Computer Applications, vol. 2, no. 9, pp. 24-31, June, 2010 published by Foundation of Computer Science.
DOI: 10.5120/694-975.
URI: http://www.ijcaonline.org/archives/volume2/number9/694-975.

[2] Sameer Singh Chauhan and R. C. Joshi, ***"A Weighted Mean Time Min-Min Max-Min Selective Scheduling Strategy for Independent Task Scheduling on Grid"***, In Proceedings of 2nd IEEE International Advance Computing Conference - 2010, pp. 4-9, February, 2010.

DOI: 10.1109/IADCC.2010.5423047

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5423047

&isnumber=5422877

[3] Sameer Singh Chauhan and R. C. Joshi, ***"Resource Performance based Grid Task Scheduling"***, 4th International Conference on Information Processing – 2010, August 6-8, 2010, Bangalore, India. (Accepted)

[4] Sameer Singh Chauhan and R. C. Joshi, ***"A Heuristic for QoS Based Independent Task Scheduling in Grid"***, IEEE International Conference on Industrial and Information Systems – 2010 (ICIIS 2010), July 29 – August 1, 2010, NIT Karnataka, Surathkal, Karnataka, India. (Accepted)

[5] Sameer Singh Chauhan and R. C. Joshi, ***"Multiple QoS Guided Heuristic Algorithm for Independent Task Scheduling in Grid"***, International Conference on Advances in Information and Communication Technologies – 2010 (ICT 2010), Kochi, India. (Accepted, proceedings will be published by Springer LNCS-CCIS).