Efficient Scheduling Heuristics for Independent Tasks in Computational Grids

Sanjaya Kumar Panda

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela - 769008, Odisha, India

Efficient Scheduling Heuristics for Independent Tasks in Computational Grids

Thesis submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

by

Sanjaya Kumar Panda

(Roll No.: 211CS2285)

under the supervision of

Prof. Pabitra Mohan Khilar

Assistant Professor

Department of Computer Science and Engineering, NIT Rourkela

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela - 769008, Odisha, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela - 769008, Odisha, India.

Rourkela
22^{nd} May 2013

Certificate

This is to certify that the work in the thesis entitled ***Efficient Scheduling Heuristics for Independent Tasks in Computational Grids*** by ***Sanjaya Kumar Panda***, bearing ***Roll Number 211CS2285***, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of ***Master of Technology*** in ***Computer Science and Engineering*** with specialization in ***Information Security***. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. Pabitra Mohan Khilar

Acknowledgment

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis. Foremost, I would like to express my sincere gratitude to my supervisor Prof. Pabitra Mohan Khilar for providing me with a platform to work on challenging areas of grid computing and fault tolerance. His valuable comments and suggestions were encouraging. He was the one who showed me the path from the beginning to end.

I am also thankful to Prof. Santanu Kumar Rath, Prof. Sanjay Kumar Jena, Prof. Banshidhar Majhi, Prof. Durga Prasad Mohapatra, Prof. Ashok Kumar Turuk, Prof. Bibhudatta Sahoo and Prof. Manmath Narayan Sahoo for giving encouragement and sharing their knowledge during my thesis work.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the Department of Computer Science and Engineering who have been kind enough to help us in their respective roles.

I would like to thank all my friends, lab mates and research scholars for their encouragement and understanding. They made my life beautiful and helped me every time when I was in some problem.

Most importantly, none of this would have been possible without the love and patience of my family. My family, to whom this thesis is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to them.

Sanjaya Kumar Panda

Abstract

Grid computing is an extension to parallel and distributed computing. It is an emerging environment to solve large scale complex problems. It enables the sharing, coordinating and aggregation of computational machines to fulfil the user demands. Computational grid is an innovative technology for succeeding generations. It is a collection of machines which is geographically distributed under different organisations. It makes a heterogeneous high performance computing environment. Task scheduling and machine management are the essential component in computational grid. Now a day, fault tolerance is also playing a major role in computational grid. The main goal of task scheduling is to minimize the makespan and maximize the machine utilisation. It is also emphasis on detection and diagnosis of fault. In computational grid, machines may join or leave at any point of time. It may happen that machine is compromised by an advisory or it may be faulty due to some unavoidable reason like power failure, system failure, network failure etc. In this thesis, we address the problem of machine failure and task failure in computational grid. Also, we have focused on improving the performance measures in terms of makespan and machine utilisation. A simulation of the proposed heuristics using MATLAB is presented. A comparison of our proposed heuristics with other existing heuristics is conducted. We also demonstrate that number of task completion increases even if some of the machine work efficiently in computational grid.

***Keywords*:** Computational Grid, Batch Mode, Independent Task, Task Scheduling, Makespan, Quality of Service, Fault Tolerance

Acronyms

BOINC	Berkeley Open Infrastructure for Network Computing
PC	Personal Computer
GMB	Grid Machine (or Resource) Broker
GRS	Grid Referral Service
IBM	International Business Machines
SSI	Single System Image
SETI	Search for Extra Terrestrial Intelligence
NSF	National Science Foundation
NASA	National Aeronautics and Space Administration
LHC	Large Hadron Collider
GPU	Graphical Processing Unit
TQ	Task Queue
MET	Minimum Execution Time
LBA	Limited Best Assignment
UDA	User Directed Assignment
FCFS	First Come First Served
EET	Expected Execution Times
MCT	Minimum Completion Time
OLB	Opportunistic Load Balancing
KPB	K - Percent Best
SA	Switching Algorithm
WMTG-min	Weighted Mean execution Time Guided-minimum
WMTSG-min	Weighted Mean execution Time Sufferage Guided-minimum
RASA	Resource Aware Scheduling Algorithm
LBMM	Load Balanced Min-Min
QoS	Quality of Service
GIS	Grid Information Service
DQ	Difference Queue
TEQ	TEmporary Queue
IRCTC	Indian Railway Catering and Tourism Corporation
CPU	Central Processing Unit
RAM	Random Access Memory

ROM	Read Only Memory
SV	Sufferage Value
MU	Machine (or Resource) Utilisation
AMU	Average Machine (or Resource) Utilisation
RSA	Rivest Shamir Adleman
SMPS	Switched Mode Power Supply
RT	Ready Time
ECT	Expected Completion Time
RET	Remaining Execution Time

Notations

m	Total number of tasks (or meta-tasks)
n	Total number of machines (or resources)
T_i	Task ID of task i
M_j	Machine ID of machine i
S	A scheduling strategy
$E_{i,j}$	Execution time for task i on machine j
$M(S)$	Makespan of scheduling strategy S
$M(S_{M_j})$	Makespan of machine j using scheduling strategy S
$MU(S)$	Machine (or resource) utilisation of scheduling strategy S
$MU(S_{M_j})$	Machine (or resource) utilisation of machine j using scheduling strategy S
$F(S)$	Completion time of scheduling strategy S
$F(S_{T_i})$	Completion time of task i using scheduling strategy S
$I(S)$	Idle time of scheduling strategy S
$I(S_{M_j})$	Idle time of machine j using scheduling strategy S
$E(S)$	Total execution time of scheduling strategy S
$E(S_{T_i})$	Execution time of task i using scheduling strategy S
$T_i \rightarrow M_j$	Task i is scheduled to M_j
$T_i \not\rightarrow M_j$	Task i is not scheduled to machine j
C	Completion Time
$C_{i,j}$	Completion Time for task i on machine j
R_j	Ready time of machine j

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The computational power of a single computer cannot provide sufficient power to run large scale complex problems [38]. It can be enhanced by improving the speed, memory, processor and many more. Even if it has speedily increased up to some extent, still some future improvements are required. Alternatively, it is possible to connect more than one computer together to achieve large computational powers. The collection of independent computers that provides the user as a single system image is called distributed system. The computers are independent of each other and it is under different organisations. The computers may join or leave at any point of time. Grid computing is an example of a distributed system. The grid was coined by Ian Foster and Carl Kesselman in the mid 1990s [1]. The purpose of the grid computing is to provide computational power to solve large scale complex problems. It is heterogeneous structure. It means each participating computer in the grid may have different specifications. The machines are distinguished in many ways like reliability, availability, scalability, performance and fault tolerance. It depends on the user requirements to assign the machines according to the problem. There is always a trade off between the machine specifications. For example, a machine may be available for 24 hours but gives poor performance or a machine may be available for very few hours but gives high performance.

Grid computing provides flexibility to solve a very large problem in a single computer. Also, it provides a multi-user environment. Multi-user environment offers the user to participate in the grid project and use the computer for personal propose at the same time. For example, $BOINC$ project [2]. In this project, we can participate in any number of projects without interfering our personal work.

The grid computing environment consists of PCs, workstations, clusters, servers and supercomputers [51]. This environment has various entities like grid user, machine, GMB, GIS, input, output and many more. The grid users or producers or machine owners have responsibility to satisfy the end user or consumer demands. To fulfil the user demands, GMB leases or hires grid machines and services based on cost, efficiency, capability and availability [62]. Both producer and consumer are distributed geographically with different standard time zones [3] [36] [39] [40] [58].

1.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

In Section 1.2, an introduction to grid computing, types of grid and characteristics of computational grids are discussed.

In Section 1.3, the real life grid projects are discussed.

In Section 1.4, the grid architecture proposed by Buyya et al. is presented. The four layer architecture and its functionality is briefly discussed [13].

In Section 1.5, the different types of grid faults are discussed.

In Section 1.6, the nature of scheduling is discussed.

Applications of grid computing, objective and motivation is presented in Section 1.7, Section 1.8 and Section 1.9 respectively.

1.3 Grid Computing

Grids are widely used for high performance computing applications because of the high cost of massively parallel processors and the wide availability of network workstations [4]. It enables sharing, aggregation, monitoring, pooling and selection of data and computational machines [3] [43] [44] [45]. A computational grid acts like a virtual organisation consisting of heterogeneous machines. A virtual organisation consists of a set of individuals or institutions or providers. They are defined by a sharing rule like what is shared, who is allowed to share, who is allowed to view the content, what is the boundary of sharing and the conditions under which the sharing takes place [5]. In most organizations or institutions, the computing machines are idle most of the time. Alternatively, the machines are not utilised properly.

The easiest use of the grid is to create a replica of tasks and run it on several machines [57]. The machine on which some tasks are running might be busy. So, the execution of the later tasks is delayed till the previous tasks are served. By creating a replica of tasks, the task can be run on an idle machine.

Let us consider a banking system or an institution, If a cashier or a staff works for seven hours per day than the total work period in a week is forty two hours. But, there are 168 hours per week. So, the machine utilisation is only 25%. So, machines are under utilized. The rest 75% can be used for other works like to participate in $BOINC$ projects. In a IBM case study in 2000, it is mentioned that the average utilisation rate is 5 to 10% for PCs and 30 to 35% for servers [6]. The observation carried out are true even today [7]. Computational grid provides a way to explore these idle machines and increase the efficiency of the machine.

Scheduling and machine management is the key component of a grid system [54]. These components are responsible for fulfilling the user requirements. However, GMB is responsible for mapping the jobs to the available machines [60]. Also, it finds the available machine list from GRS. It splits the job into a number of small units and each unit is distributed to a machine. At last, it combines the results from different machines and get back to the user. But, the user has no knowledge of the distributed machines. It has submitted the job to the single system and gets the results from that system only. This property is called as SSI.

Message passing interface and parallel virtual machines allow the network of heterogeneous machines to get a large amount of computational power and memory. It allows the programmers to write parallel programs [7].

1.3.1 Types of Grid

There are different types of grid used for different applications. They are based on two factors: functionality and scale.

Types of Grid on Basis of Functionality

- **Computational Grid:** It is a collection of machines in a network that is used to solve a particular problem at the same time.
- **Data Grid:** It gives an environment to support data selection, data storage and data manipulation of huge amounts of data stored in different systems [52].
- **Collaborative Grid:** It is used to solve a problem by multiple organisations to get a common benefit. For example, users from different domains are working on different components of a $BOINC$ project without disclosing the technologies [8] [48].

• **Network Grid:** It gives a fault-tolerant, high speed communication and reliable services [8].

• **Utility Grid:** It is not only shared computational power and data but also share software and special equipments [8].

Types of Grid on Basis of Scale

• **Cluster Grid:** It is homogeneous structure. It provides services to the group or departmental level. The number of machines is in between 2 to 100. They are connected by system area network or local area network [48].

• **Enterprise Grid:** It is heterogeneous structure. It provides services to the multiple groups or multiple departments or organisational level. The number of machines is many 100s.

• **Global Grid:** It is also heterogeneous structure. It is the collection of multiple enterprise grids. It provides services to the national or international level. The number of machines is many 1000s or millions.

1.3.2 Characteristics of Computational Grids

The characteristics of computational grids are described as follows:

• **Machine Configuration:** There are two types of machine configuration: homogeneous and heterogeneous. In homogeneous, all machines can have same operating systems, processors, speed, model, system type and memory. But, in heterogeneous, all machines can have different operating systems, processors, speed, model, system type and memory [4]. The grid is heterogeneous in nature.

• **Single System Image:** In Grid, the collection of machines is interconnected in such a fashion that appears like a unified machine. It is called as SSI. It resides between the operating system and user-level environment [4].

• **Machine Sharing:** The machines are widely distributed and may be owned by different administrative domains. It may join or leave at any point of time.

• **Scalability:** The grid machines may be ranging from a few to millions. It may leads the problems of performance degradation. So, the grid must be able to accommodate the growth.

- **Geographical Distribution:** The grid machines are distributed in different places. It is under different domains or organisations.
- **Multiple administrations:** Each domain or organisation may have different security policies like public key infrastructure under which the machines can be accessed [9] [63]. The machine may be left at any point of time if security policies does not met.

1.4 Grid Projects

Some of the grid real life projects are $SETI$@home, Milkyway@home and Einstein@home [10] [11] [12]. $SETI$@home is funded by NSF, $NASA$ [10]. These projects are running in $BOINC$ middleware systems [2]. $BOINC$ is an open source systems. The $BOINC$-based projects are categorized into following types:

1.4.1 Astronomy, Physics and Chemistry

The following projects are under astronomy, physics and chemistry categories: Asteroids@home, Constellation, Cosmology@home, eOn, Leiden Classical, LHC@home, LHC@home Test4Theory, Milkyway@home, $SETI$@home, Spinhenge@home and uFluids@home.

1.4.2 Biology and Medicine

The following projects are under biology and medicine categories: Docking@home, FightMalaria@home, GPUGrid.net, Malariacontrol.net, POEM@home, RNA World, Rosetta@home, SIMAP, Superlink@technion and The Lattice Project.

1.4.3 Cognitive Science and Artificial Intelligence

The following projects are under cognitive science and artificial intelligence categories: FreeHAL, MindModeling@home.

1.4.4 Distributed Sensing

The following projects are under distributed sensing categories: Quake Catcher Network and Radioactive@home.

1.4.5 Earth Sciences

The following projects are under earth sciences categories: Climateprediction.net and Virtual Prairie.

1.4.6 Mathematics, Computing and Games

The following projects are under mathematics, computing and games categories: ABC@home, Chess960@home, Collatz Conjecture, DistRTgen, Enigma@home, NFS@home, NumberFields@home, OProject@home, Primaboinca, PrimeGrid, SAT@home, SubsetSum@home, Sudoku@vtaiwan, Surveill@home, SZTAKI Desktop Grid, VolPEx and VTU@home.

1.4.7 Multiple applications

The following projects are under multiple application categories: CAS@home, EDGeS@home, Ibercivis

1.5 Grid Architecture

Grid architecture is the art that identifies the components and its relation with each other. It consists of four layers: fabric, core middleware, user-level middleware and applications and portals layers [13]. Each layer constitutes the services offered by the lower layer and provides some services at the same layer. The architecture in Buyya et al. is shown in Figure 1.1 [13].

1.5.1 Grid Fabric

This layer consists of distributed machines like computers, networks, storage systems, data sources and scientific instruments. The machines are in the form of clusters of PCs or piles of PCs, supercomputers, servers or workstations and ordinary PCs which run on different platforms. Scientific instruments like a seismograph (for recording earthquake), seismometer (for measuring earthquake intensity), seismoscope (for detecting earthquake), telescope and sensor networks give real time data that can be stored in a storage system and transmitted to computational sites [13].

1.5.2 Core Middleware

This layer consists of distributed machine services like security, QoS, trading and process management. This layer hides the complexity and heterogeneity of the lower level (i.e. fabric level) by giving a consistent method for accessing the machines [13].

1.5.3 User-level Grid Middleware

This layer consists of compilers, libraries, debuggers and web tools. It utilizes the interfaces provided by lower level middleware (i.e. core middleware) to provide higher levels of abstractions [13]. Machine broker is responsible for managing, selecting and scheduling the tasks on machines.

1.5.4 Applications and Portals

Grid application includes scientific, engineering applications. It is developed grid enabled programming environments and interfaces. For example, a challenging problem like milkyway@home requires computational power, access to remote data and may need to interact with scientific instruments. Grid portals offer web enabled applications in which users can submit the tasks and get back the results from remote machines [13].

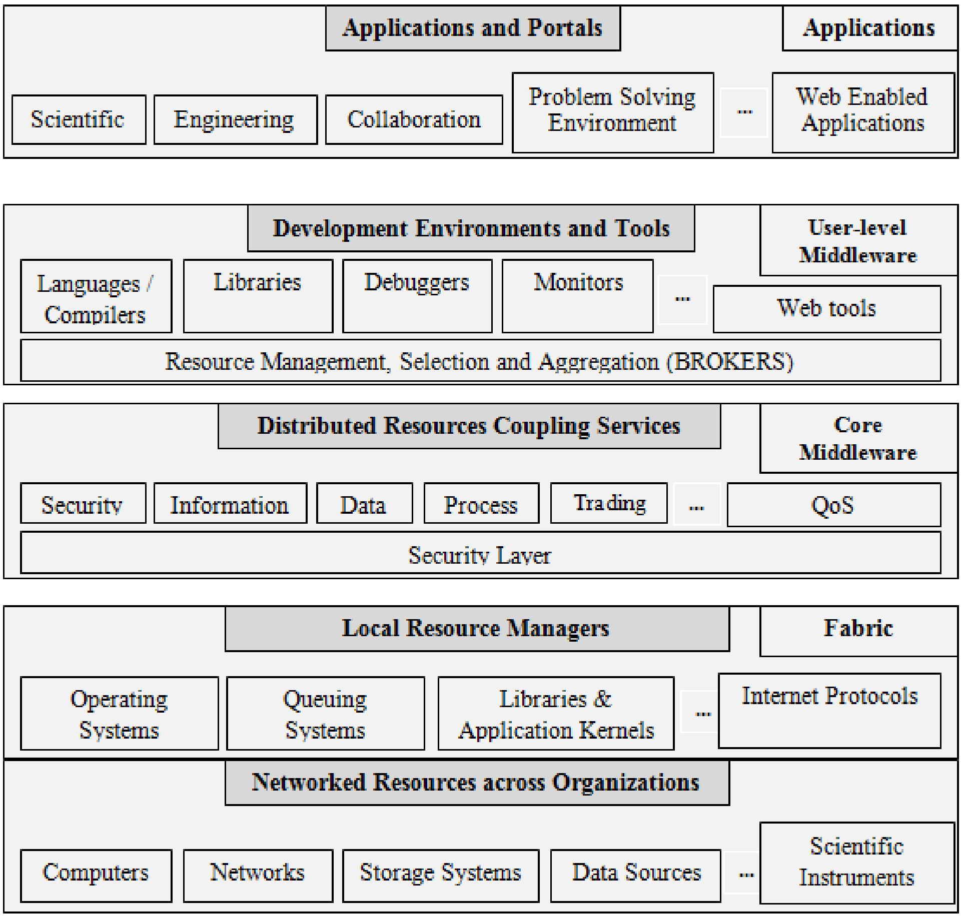

Figure 1.1: A Layered Grid Architecture and Components

1.6 Fault

A fault is an abnormal condition or unexpected behavior in the machine. In the grid, a machine can be behave abnormally due to various reasons like hardware fault, software fault, application fault and many more. It is important to detect, manage and recover the fault in time irrespective of user intervention. The grid user should get the scheduling result even if the fault exists.

There are various types of faults in the grid. They are:

1.6.1 Hardware Fault

It may occur due to faulty hardware components like CPU, RAM, ROM, $SMPS$, cache, hard disk and motherboard. Hardware fault rates are low and still decreasing [14]. One of the main reasons behind the hardware fault is violating the hardware specification. For example, a computer system is designed to work on 220V to 240V AC power supply. Otherwise, it is prone to failure. The variation in the power supply may lead to hardware component failure.

1.6.2 Software Fault

It may occur due to an unhandled exception like array index out of bound, divided by zero, invalid input and specifying an incorrect path of a file, data or memory etc.

1.6.3 Network Fault

It may occur due to connection fault, machine fault and packet loss. As machines are distributed geographically, network faults are more obvious. Packet loss may cause due to machine out of order, network congestion and link breakage. A packet may be corrupted in transmission because of network problems [64].

1.6.4 Application and Operating System Fault

It may occur due to specific application problems like memory leak and operating system problem like deadlock, improper machine management, dynamic link library problem and program crashing problem [14].

1.6.5 Response Fault

It may occur due to a lower level fault, slow connection and faulty processor. The system gives an arbitrary result which may or may not be correct. Alternatively, it oscillates in between correct and incorrect result.

1.7 Task Scheduling in Grid

Task scheduling is the dynamic mapping of a group of independent tasks into the distributed machines. It has two steps: matching and scheduling [15] [37]. Matching is the mapping between the tasks and the machines. Scheduling is the execution order of the tasks. In this thesis, the heuristics are non-preemptive in nature and assumed that the tasks have no priorities or deadlines. Mapping the tasks onto the distributed machines is an NP-complete problem [15] [16] [17] [18] [50] [53] [61].

There are different types of scheduling in grid used for different applications. They are listed below.

1.7.1 Immediate versus Batch Mode Scheduling

In immediate mode, the tasks is computed one after another. Alternatively, the task arrives first in TQ, will be computed first. Even if, more than one tasks are arrived at a time, this mode takes one task at a time and selects the first one rather than the best one. In batch mode, a batch of tasks arrives at a time. One of the task is selected from the batch of the tasks. Alternatively, the task arrives first in TQ, may or may not be computed first. If the batch size is one, then batch mode heuristics acts like immediate mode heuristic.

1.7.2 Static versus Dynamic Scheduling

In static scheduling, once the matching phase is over, the GMB cannot interfere in scheduling phase. New tasks cannot be joined in the middle of computations. So, a high priority task cannot be processed at the scheduled time. The deadline based tasks may not be processed before the deadline. In dynamic scheduling, the tasks are arrived in between the computation. The GMB can alter the scheduling sequence. New task may be participating in the middle of computations. A high priority task may be processed in scheduled time. Also, the deadline based tasks may be processed before the deadline.

1.7.3 Non-preemptive versus Preemptive Scheduling

In non-preemptive scheduling, once a task is assigned to a machine, it cannot be released before the completion. A deadline based task has to wait until the computation is over even if it misses the deadline. In preemptive scheduling, a task may be released before the completion is over. When a high priority task arrives, the current task checks the priority. If the current task priority is low, then it releases the machine. Otherwise, it continues the computation.

1.8 Applications of Grid Computing

1. $SETI$@home
2. distributed.net in 1977 has been applied a method to crack RSA 64-bit encryption algorithm. The task was completed on 14^{th} July 2002 using more than 3,00,000 machines over 1757 days [42].
3. folding@home project (from stanford university) is used to solve large scale molecular simulations [42].

4. climateprediction.net project (from oxford university) is used to predict the weather climate throughout the world [42].

5. Enabling Grids for E-sciencE project

6. Distributed European Infrastructure for Scientific Applications projects

7. UKs National Grid Service

1.9 Objective

The main objectives we find from the motivation to work in scheduling are discussed as follows:

- **Scheduling Problem**: To design an efficient scheduling heuristic by which the makespan is minimised and machine utilisation is increased.

- **Fault Problem**: To design an efficient scheduling heuristic which deals with the machine and task failure.

1.10 Motivation

Computational grids are used widely for executing large scale applications with computation intensive problems in science and engineering. Braun et al. presented an extensive survey on eleven static heuristics for mapping independent tasks onto a distributed computing system [17]. Maheswaran et al. proposed two immediate mode and one batch mode heuristics for mapping independent tasks onto distributed computing system [15]. Xiaoshan et al. introduced QoS guided heuristics for mapping independent tasks [18]. Amudha et al. introduced QoS priority based scheduling for mapping independent tasks [59]. Xhafa et al. simulated all immediate mode and batch mode heuristic in C++ [19] [20]. Apart from that, a new batch mode heuristic called longest job to fastest resource - shortest job to fastest resource heuristic was introduced. Chaturvedi et al. implemented ten static heuristics for mapping independent tasks and a new mode of heuristic was introduced [21]. In scheduling, authors are not paying that much attention towards skew data and fault tolerance for mapping independent tasks. It may lead to serious performance degradation interns of makespan and machine utilisation. Some authors have proposed fault tolerance scheduling based on the fault occurrence history strategy [22] [23].

In this thesis, we proposed efficient scheduling heuristics for skew data set and fault tolerance to solve the problems mentioned above.

1.11 Thesis Organisation

The organisation of the rest of the thesis and a brief outline of the chapters is as follows.

In chapter 2, some basic concepts of scheduling and its natures have been discussed.

In chapter 3, some related work on immediate mode scheduling, batch mode scheduling, *QoS* batch mode scheduling and fault tolerance scheduling heuristics have been discussed.

In chapter 4, we have presented our proposed approaches for task scheduling in computational grid with scheduling model, the architecture of the task scheduler, timeline of the task scheduling sequence and the scheduling heuristics.

In chapter 5, we have presented our proposed approaches for fault tolerance scheduling in computational grid with scheduling model, timeline of the fault tolerance scheduling sequence and the scheduling heuristics.

In chapter 6, we focus on the implementation and experimental results. The evaluation strategies are introduced in this chapter and a comparison study of the proposed heuristics with other heuristics is provided.

Finally, chapter 7 is given to the conclusion and future work.

1.12 Summary

In this chapter, we have discussed briefly about the grid computing, various grid projects, the grid architecture, different types of fault and applications. Also, we have discussed about the different mode of scheduling and the its processing criteria.

Chapter 2

Basic Concepts

In this chapter, we discuss a few basic concepts based on which our approach has been developed.

2.1 Introduction

The main key components of scheduling is the task and the machine. The mapping of both components are represented in matrix form. The matrix is a two dimensional array, arranged in rows and columns. The row indicates the task and the column indicates the machine. Each element in the matrix represents an execution time of a task on a machine.

2.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

The different types of task is discussed in Section 2.2. The different functionality of machine is presented in Section 2.3. The different types of matrices are discussed in Section 2.4.

2.3 Task

A task is a set of instructions or data. Instruction is measured in millions instruction unit and data are measured in megabytes or megabits. The task may have low or high heterogeneity. In the grid, task is of two types: independent and dependent. The complete hierarchy of task is shown in 2.1.

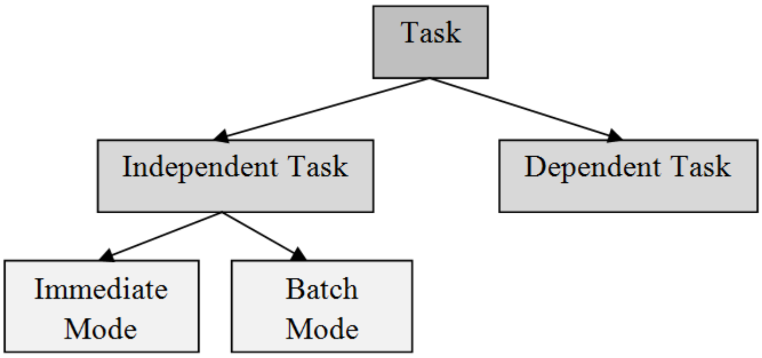

Figure 2.1: Hierarchy of Task

2.3.1 Independent Task

Independent task has no relationships between each others. Let us consider the task T_i and the task T_j that has independent of each others. So, the scheduling sequence does not affect the computations. Alternatively, the tasks are scheduled in two ways: T_i followed by T_j and T_j followed by T_i. Independent tasks are represented in matrix form. The tasks that do not have any dependency among each others are referred as Meta tasks [41] [56].

Independent tasks are scheduled in two ways: immediate and batch mode. In immediate mode, tasks are scheduled as soon as it arrives. In batch mode, tasks are scheduled in a batch.

2.3.2 Dependent Task

Dependent task has a relationship between each others. Let us consider the task T_i and the task T_j that has dependent of each others i.e. the task T_j is dependent on the task T_i. So, the scheduling sequence will affect the computations. Alternatively, the tasks are scheduled in only one way: T_i followed by T_j. Dependent tasks are represented in directed acyclic graph form or task graph form.

2.4 Machine

Machine is the producer or service in the grid. It is distributed geographically and it is under different organisations or institutions or domains. It may participate or leave at any point of time from the grid. Each machine may have different security policies or guidelines. It provides different functionality like reliability, availability, scalability, performance and fault tolerance. According to user functional requirements, the scheduler assigns the tasks to the machines.

2.5 Types of Matrix

There are three types of matrices: consistent, inconsistent and semi-consistent [7].

2.5.1 Consistent Matrix

A matrix is said to be consistent if and only if a machine M_i takes earliest execution time to execute a task T_i than machine M_j, then the machine M_i always takes earliest execution time to execute any task T_i than machine M_j. It can be mathematically expressed as follows: Let us consider the EET matrix shown in Equation (2.1).

Here, each row indicates a task and each column indicates a machine.

$$\begin{pmatrix} E_{1,1} & E_{1,2} & ... & E_{1,n-1} & E_{1,n} \\ ... & ... & ... & ... & ... \\ E_{m,1} & E_{m,2} & ... & E_{m,n-1} & E_{m,n} \end{pmatrix} \tag{2.1}$$

Assume that, $E_{1,1} < E_{1,2} < ... < E_{1,n-1} < E_{1,n}$

then $\forall_i(E_{i,1} < E_{i,2} < ... < E_{i,n-1} < E_{i,n})$ are true.

where $i =$ any task T_i ranges from 1 to m

2.5.2 Inconsistent Matrix

A matrix is said to be inconsistent if and only if a machine M_i takes earliest execution time to execute a task T_i than machine M_j, then the machine M_i may or may not takes earliest execution time to execute any task T_i than machine M_j. The machine M_i may be faster for some tasks and slower for rest. It can be mathematically expressed as follows: Let us consider the EET matrix shown in Equation (2.1).

Assume that, $E_{1,1} < E_{1,2} < ... < E_{1,n-1} < E_{1,n}$

then it is not necessary that $\forall_i(E_{i,1} < E_{i,2} < ... < E_{i,n-1} < E_{i,n})$ are true.

where $i =$ any task T_i ranges from 1 to m

2.5.3 Semi-consistent Matrix

A matrix is said to be semi-consistent if and only if a sub matrix is consistent. It can be mathematically expressed as follows: Let us consider the EET matrix shown in Equation (2.1).

Assume that, $E_{1,1} < E_{1,2} < ... < E_{1,n-1} < E_{1,n}$

then $\forall_i(E_{i,j} < E_{i,j+k} < ... < E_{i,j+k1} < E_{i,j+kx})$ are true.

where $1 \leq j \leq m$, $i =$ any task T_i ranges from 1 to m,

$j < j + k < j + k1 < j + k2 < ... < j + kx$,

$k < k1 < k2 < ... < kx$

2.6 Summary

In this chapter, we have discussed briefly about the task, the machine and various types of matrix. Also, we have discussed the nature of each matrix.

Chapter 3

Related Work

In this chapter, we will provide a brief literature survey of existing scheduling heuristics with merits and demerits.

3.1 Introduction

Researchers have proposed various heuristics based on different criteria. The works are categorized into two types: immediate and batch mode heuristic. Again, each mode of heuristic is applied to three types of matrices: consistent, inconsistent and semi-consistent. The batch mode heuristics are categorized into two types: QoS and non-QoS.

3.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

Related work on immediate mode, batch mode and QoS batch mode and fault tolerance scheduling is discussed in Section 3.2, Section 3.3, Section 3.4 and Section 3.5 respectively.

3.3 Related Work on Immediate Mode Scheduling Heuristics

In this section, five immediate mode heuristics are explained. These are:

3.3.1 MET

It is also called as LBA and UDA [15]. It assigns each task to the machine that gives the least amount of execution time. Also, it assigns each task to the machine in $FCFS$ basis. The least execution time taken machine is fully overloaded and other machines are completely idle in consistent type of matrices because, it is not considering machine ready time. This heuristic requires $O(n)$ time to assign each task to the machine [15] [24].
Merits: It is very simple and inexpensive.
Demerits: Load imbalance

It can be mathematically expressed as follows:

Let us consider the EET matrix shown in Equation (2.1). The EET of task T_i can be calculated as shown in Equation (3.1).

$$T_i \longrightarrow min(E_{i,1}, E_{i,2}, E_{i,3}, ..., E_{i,n}) \quad (3.1)$$

3.3.2 MCT

It assigns each task to the machine that gives the earliest completion time. Also, it assigns each task to the machine in $FCFS$ basis like MET [47]. The completion time can be calculated as shown in Equation (3.2). The ready time of the machine is the time required for the machine to complete all assigned tasks to it. This heuristic requires $O(n)$ time to assign each task to the machine [15] [24].

$$Completion\ time = Execution\ time + Ready\ time \tag{3.2}$$

Merits: It is an improvement over MET. Load imbalance is reduced to some extent.
Demerits: It requires the ready time as an extra parameter.

It can be mathematically expressed as follows: Let us consider the EET matrix shown in Equation (2.1). The EET of task T_1 can be calculated as shown in Equation (3.3).

$$T_1 \longrightarrow min(E_{1,1}, E_{1,2}, E_{1,3}, ..., E_{1,n}) \tag{3.3}$$

Let $T_1 \longrightarrow M_\alpha$then the execution time of the task T_1 on machine M_α is $E_{1,\alpha}$ So, the expected completion time of the task T_2 can be calculated as shown in Equation (3.4). Here, $E_{1,\alpha}$ is the ready time of machine α.

$$T_2 \longrightarrow min(E_{2,1} + E_{1,\alpha}, E_{2,2} + E_{1,\alpha}, E_{2,3} + E_{1,\alpha}, ..., E_{2,n} + E_{1,\alpha}) \tag{3.4}$$

$$\text{where } E_{1,\alpha} = \begin{Bmatrix} 1 & if\ T_i \longrightarrow M_\alpha \\ 0 & Otherwise \end{Bmatrix}$$

3.3.3 OLB

It assigns a task to the machine that becomes idle next. It is not taking the execution time of the task and completion time of the task into consideration. This heuristic requires $O(n)$ time to assign each task to the machine [15] [16].
Merits: It is very simple and inexpensive [46].
Demerits: Execution time of the task is not considered.

It can be mathematically expressed as follows: Let us consider the RT matrix shown in Equation (3.5). The task T_1 is assigned to the least ready time machine as shown in Equation (3.6). The EET of task T_1 can be calculated as shown in Equation (3.7).

$$(R_1 \quad R_2 \quad R_3 \quad ... \quad R_n) \tag{3.5}$$

$$T_1 \longrightarrow min(R_1, R_2, R_3, ..., R_n) \tag{3.6}$$

$$T_1 \longrightarrow E_{1,1} + R_1, E_{1,2} + R_2, E_{1,3} + R_3, ..., E_{1,n} + R_n \tag{3.7}$$

$$\text{where } R_i = \left\{\begin{smallmatrix} 1 & T_1 \longrightarrow M_i \\ 0 & Otherwise \end{smallmatrix}\right\}$$

3.3.4 KPB

It assigns each task to the machine based on the value of K. It chooses a subset of machines (n') from the available machines. The (n') depends on the value of n and K. The (n') can be calculated as shown in Equation (3.8). At last, it assigns each task to the machine that gives earliest completion time from the K machines. KPB heuristic acts like MCT heuristic when $K = 100$ and it acts like MET heuristic when $K = 100/n$. The heuristic selection is shown in Equation (3.9). If $K = 100$, then the (n') is same as n. If $K = 100/n$, then (n') is a proper subset of n. KPB heuristic requires $O(n \ log \ n)$ time to assign each task to the machine [15].

Merits: It takes less time to assign each task.

Demerits: It depends on the value of K. If $K = 100/n$ then it may lead to the load imbalance problem (consistent matrix).

$$(n') = n \times (K/100) \tag{3.8}$$

$$\text{where } (n') \subseteq n$$

$$Heuristic = \begin{Bmatrix} MET & ifK = 100/n \\ MCT & ifK = 100 \\ KPB & Otherwise \end{Bmatrix} \tag{3.9}$$

3.3.5 SA

It is a hybrid heuristic based on MET and MCT. Let r_{max} is the maximum ready time of all available machines; r_{min} is the minimum ready time of all available machines and π is the load balance index. The value of π can be calculated as shown in Equation (3.10). The value of π is in between 0 to 1. The initial value of π is 0. This heuristic uses two threshold values: π_l (low load balance index) and π_h (high load balance index). Note that $0 < \pi_l < \pi_h < 1$. It starts with MCT heuristic and continue task mapping. When the value of π is reached to π_h or above, it uses MET heuristic to decrease the load balance factor. If the value of π is reached to π_l or below then it uses MCT heuristic to increase the load balance factor. This heuristic gives optimum makespan value when $\pi_l = 0.6$ and $\pi_h = 0.9$. It requires $O(n)$ time to assign each task to the machine [15].

$$\pi = r_{min}/r_{max} \tag{3.10}$$

Merits: It gives the makespan value in between MET and MCT for consistent and semi-consistent matrices [20].
Demerits: It is very difficult to choose the optimum value π_l and π_h in each data set.

3.4 Related Work on Batch Mode Scheduling Heuristics

In this section, nine batch mode heuristics are explained. These are:

3.4.1 Min-Min

It is a hybrid heuristic based on MET and MCT immediate mode heuristics. Let us consider the EET matrix shown in Equation (2.1). It chooses a machine for each task that provides earliest completion time. The resultant matrix is a column matrix as shown in Equation (3.11). Again, it chooses an earliest completion time from the column matrix as shown in Equation (3.12). Let task T_i takes earliest completion time in Equation (3.12) where i be the any value from 1 to m, depends on the min function. Then, this heuristic assigns task T_i to the machine that gives earliest completion time. If the number of long tasks is more than the number of short tasks, then the min-min heuristics gives optimum makespan value than the max-min heuristic (Section 3.2.2) [15] [24]. Alternatively, if the completion times of tasks are positively skewed, then min-min gives optimum value than max-min heuristic. It requires $O(m^2n)$ time to assign the tasks to the machines [15] [24].

$$\begin{pmatrix} E_{1,\alpha} \\ E_{2,\beta} \\ E_{3,\gamma} \\ ... \\ E_{m,v} \end{pmatrix} \tag{3.11}$$

$$\begin{aligned} \text{where } E_{1,\alpha} &= min(E_{1,1}, E_{1,2}, E_{1,3}, ..., E_{1,n}) \\ E_{2,\beta} &= min(E_{2,1}, E_{2,2}, E_{2,3}, ..., E_{2,n}) \\ E_{3,\gamma} &= min(E_{3,1}, E_{3,2}, E_{3,3}, ..., E_{3,n}) \\ &\text{...} \\ E_{m,v} &= min(E_{m,1}, E_{m,2}, E_{m,3}, ..., E_{m,n}) \end{aligned}$$

$$T_i \rightarrow min(E_{1,\alpha}, E_{2,\beta}, E_{3,\gamma}, ..., E_{m,v}) \tag{3.12}$$

$$\text{where } i = 1 \;\; or \;\; 2 \;\; or \;\; ... \;\; or \;\; m$$

3.4.2 Max-Min

It is also a hybrid heuristic based on MET and MCT immediate mode heuristics. Let us consider the EET matrix shown in Equation (2.1). It chooses a machine for each task that provides earliest completion time. The resultant matrix is a column matrix as shown in Equation (3.11). Again, it chooses a latest completion time from the column matrix as shown in Equation (3.13). Let task T_i takes latest completion time in Equation (3.12) where i be the any value from 1 to m, depends on the max function. Then, this heuristic assigns task T_i to the machine that gives earliest completion time [49]. If the number of long tasks is less than the number of short tasks, then the max-min heuristics gives optimum makespan value than the min-min heuristic [15] [24] [35]. Alternatively, if the completion times of tasks are negatively skewed, then the max-min gives optimum value than the min-min heuristic. It requires$O(m^2n)$ time to assign the tasks to the machines [15] [24].

$$T_i \rightarrow max(E_{1,\alpha}, E_{2,\beta}, E_{3,\gamma}, ..., E_{m,v}) \tag{3.13}$$

$$\text{where } i = 1 \ \ or \ \ 2 \ \ or \ \ ... \ \ or \ \ m$$

3.4.3 Sufferage

This heuristic assigns the tasks to a machine based on sufferage value. Sufferage value is the difference between the second earliest completion time and first earliest completion time. It is shown in Equation (3.14). A task that suffers most is assigned to a machine first. Let sufferage value of the task T_i and the task T_j is S_1 and S_2 respectively. Assume that the task T_i is already assigned to a machine M_i and the task T_j is going to assign to the machine M_i. Then, this heuristic finds the status of the machine i.e. either assigned or unassigned. According to the above situation, it is assigned to the task T_i. So, S_1 (T_i) and S_2 (T_j) are compared. If S_1 (T_i) $<$ S_2 (T_j), then unassigned the task T_i from the machine M_i and assign the task T_j to the machine M_i. The task T_i is scheduled in the next iteration. It requires $O(S^2n)$ time to map a task of size S [15].

$$Sufferage\,Value = Second\,earliest\,completiontime - First\,earliest\,completion\,time \tag{3.14}$$

3.4.4 Duplex

It is a hybrid heuristic based on min-min and max-min. It performs both the heuristics and uses the optimum solution. It is preferable in which min-min or max-min gives optimum solution [15].

3.4.5 WMTG-min

It assigns the task to a machine that has maximum weighted mean execution time. Let us consider the EET matrix shown in Equation (2.1). At first, it finds the average execution time of each machine. It is shown in Equation (3.15). Next, it finds the sum of average execution time. Let w_j is the performance metric of the machine M_j. It can be calculated using Equation (3.16). At last, we calculate the weighted mean execution time (e_i) as shown in Equation (3.17). It finds the task T_i that gives the maximum value of e_i [25].

$$
\begin{aligned}
Average = (&\frac{(E_{1,1} + E_{2,1} + E_{3,1} + ... + E_{m,1})}{m}, \\
&\frac{(E_{1,2} + E_{2,2} + E_{3,2} + ... + E_{m,2})}{m}, \\
&..., \\
&\frac{(E_{1,n} + E_{2,n} + E_{3,n} + ... + E_{m,n})}{m})
\end{aligned} \tag{3.15}
$$

$$w_j = \frac{Average_j}{\sum(Averageexecutiontime)} \tag{3.16}$$

$$e_i = \sum_{k=1}^{n} w_k E_{i,k} \tag{3.17}$$

3.4.6 WMTSG-min

This heuristic is an improvement of sufferage heuristic. Like $WMTG - min$, it finds the average execution time of each machine and the sum of average execution time. It also calculates the performance metric w_j. Then, it uses the sufferage heuristic to assign each task to a machine. Initially, all the machines are considered as unassigned. Then, it calculates the value of e_i as shown in Equation (3.18). Next, it finds the task T_i that gives the maximum value of e_i [25]. The task T_i finds the machine M_j and M_k that gives the first earliest completion time and second earliest completion time respectively. Sufferage value (S) can be calculated using Equation (3.14). If the machine M_j is unassigned then the task T_i is assigned to the machine M_j and the machine M_j is marked as assigned. If the machine M_j is assigned to a task T_k then sufferage value of the task T_k and the task T_i is compared. If S_1 (T_k) $< S_2(T_i)$, then unassigned the task T_k from the machine M_j and assign the task T_i to the machine M_j [25].

$$e_i = \sum_{k=1}^{n} w_k(R_i + E_{i,k}) \tag{3.18}$$

where R_i = Ready time of machine M_i

3.4.7 Selective

It is a hybrid heuristic based on min-min and max-min. Let us consider the EET matrix shown in Equation (2.1). It chooses a machine for each task that provides earliest execution time. The resultant matrix is a column matrix as shown in Equation (3.11). Assume that, the column matrix is in sorted order. Then, it finds population standard deviation (PSD) measures of dispersion using the column matrix. The PSD formula is shown in Equation (3.19). It finds a place p in the column matrix where the difference of two consecutive completion times is more than PSD. If the place p lies in the lower half i.e. $(m/2)$ then it applies min-min heuristic. Otherwise, it applies max-min heuristic. This heuristic requires $O(m^2n)$ time to assign the tasks to the machines [24].

$$PSD = \sqrt{\frac{(E_{1,\alpha} - M)^2 + (E_{2,\beta} - M)^2 + ... + (E_{m,v} - M)^2}{m}} \tag{3.19}$$

where $M = \frac{E_{1,\alpha}+E_{1,\beta}+...+E_{m,v}}{m}$

3.4.8 RASA

It is also a hybrid heuristic based on min-min and max-min. It performs the min-min heuristic when the available machine is odd. Otherwise, it performs max-min heuristic. If the first task is assigned using the min-min heuristic then second task is assigned using the max-min heuristic. This heuristic requires $O(m^2n)$ time to assign the tasks to the machines [26].

3.4.9 LBMM

It is also a hybrid heuristic based on min-min and MCT. It performs the min-min heuristic to assign each task to a machine. It finds the task T_i that gives the maximum completion time less than the makespan. Then, it reschedules the task T_i to avoid the load imbalance problem [27].

3.5 Related Work on QoS Batch Mode Scheduling Heuristics

In this section, three QoS batch mode heuristics are explained. These are:

3.5.1 QoS Guided Min-Min

QoS is different meaning in different applications. In the grid, it may be the bandwidth, speed, deadline, priority etc [28]. Generally, the tasks are divided into two levels of QoS: high QoS and low QoS. A task with the low QoS request can be scheduled to both low QoS and high QoS machines. However, a task with a high QoS request can only be scheduled to high QoS machines. This heuristic maps the tasks with high QoS request before the low QoS request. It performs the min-min heuristic on both high QoS and low QoS requests. However, it finds a machine from the set of QoS qualified machines in high QoS requests [18].

3.5.2 QoS Priority Grouping

In this heuristic, the tasks are divided into two groups. Tasks that can be executed on all available machines are included in the low QoS group. Alternatively, tasks that cannot be executed on at least one machine are included in the high QoS group. According to QoS level, it uses sufferage heuristic to assign the tasks to a machine [28].

3.5.3 QoS Sufferage Heuristic

It also divides the tasks into two groups: high QoS and low QoS. It schedules both high QoS and low QoS tasks based on sufferage heuristic [29].

3.6 Related Work on Fault Tolerance Scheduling Heuristics

Nazir et al. presented the problem of fault tolerance in the form of machine failure [30]. In this scheme, the GIS maintains a history of the fault occurrence. GMB uses the GIS history information to schedule the tasks. This scheme uses check pointing strategy to make scheduling more efficient and reliable.

Khanli et al. presented machine fault occurrence history strategy for scheduling in grid [22]. It is also maintains the history of the fault occurrence. Like Nazir et al., it uses genetic algorithm to schedule the tasks. This scheme uses check pointing strategy to make scheduling more efficient and reliable.

Priya et al. proposed task level fault tolerance [31]. The proposed approach considers retry, alternate machine, check pointing and replication task level techniques. Like Nazir et al. and Khanli et al., it uses genetic algorithm to schedule the tasks. This scheme uses check pointing strategy to make scheduling more efficient and reliable.

Upadhyay et al. proposed a fault tolerant technique based on checkpointing and passive replication [32]. Both techniques are combined using genetic algorithm to perform the scheduling.

Guo et al. introduced local node fault recovery technique for grid systems [33]. It is also given a study on grid service reliability modeling. To be more effective, it uses an ant colony optimization algorithm to perform the multi-objective scheduling.

Nanthiya et al. proposed a load balancing architecture with fault tolerance [34]. It introduced a load balancing algorithm among the machines. The algorithm has two phases. In the first phase, the machines are arranged according to the deadline and the fault tolerant factor. In the second phase, the load balancing algorithm is applied to balance the load of the machine.

3.7 Summary

In this chapter, we have discussed briefly about related work on immediate, batch mode, QoS batch mode heuristics along with merits and demerits. Also, we have discussed some related work in fault tolerance scheduling.

Chapter 4

Efficient Scheduling Heuristics in Computational Grids

4.1 Introduction

Let us assume a decentralised computational grid infrastructure with geographically distributed machines. The machines are managed, controlled and organised by different administrative domains. But, *GRS* keeps track of all information about the machines. Machines may have different specifications e.g. operating system, processor, speed, model, system type and memory [55]. Like machine, tasks are submitted from different administrative domains. The task may have different specifications e.g. deadline, scheduling policy, volume of instruction, volume of data and execution time [38].

4.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

The problem definition is presented in Section 4.2. In Section 4.3, we have presented the assumptions taken in this thesis. The scheduling model, architecture of GMB, timeline sequence and a research model of grid is presented in Section 4.4, Section 4.5, Section 4.6 and Section 4.7 respectively.

The SIM^2, TSA and RRTS heuristics is shown in Section 4.8, Section 4.9 and Section 4.10 respectively. In each Section, an illustration shows the analysis of the heuristics.

4.3 Problem Definition

In this thesis, we focus on the problem of scheduling m tasks on n machines. The aim is to minimize the overall processing time (or makespan) and utilizing the machines efficiently. To formulate the problem mathematically, let us consider T_i where $i = 1, 2, 3, ..., m$ as m independent tasks and M_j where $j = 1, 2, 3, ..., n$ as m machines. So, m tasks and n machines are of $m \times n$ order. EET for task T_i on machine M_j is $E_{i,j}$. EET for m tasks and n machines are shown in Equation (2.1). The main goal is to find an efficient scheduling strategy S, which minimizes the overall processing time and maximizes the machine utilisation.

4.4 Assumptions

In this thesis, we have considered following assumptions:

1. The tasks are nonpreemptive in nature.
2. The tasks are independent of each other.
3. The tasks have no deadlines or priorities.

4.5 Scheduling Model

The scheduling model consists of four blocks. The blocks are users, grid machine broker, grid referral service and machines. User submits the job(s) to the grid machine broker. The scheduling model is shown in Figure 4.1. The grid machine broker obtains available machine information from the grid referral service. It maps the jobs to available machines based on the scheduling strategy. Also, it splits the job into a number of small units called task. The grid referral service obtains information about the available machines. It is responsible for machine registration, machine directory management and status of the machine. It maintains the machine characteristics like operating system, processor, speed, bandwidth, model, system type, memory and processing cost. It provides information to the grid machine broker.

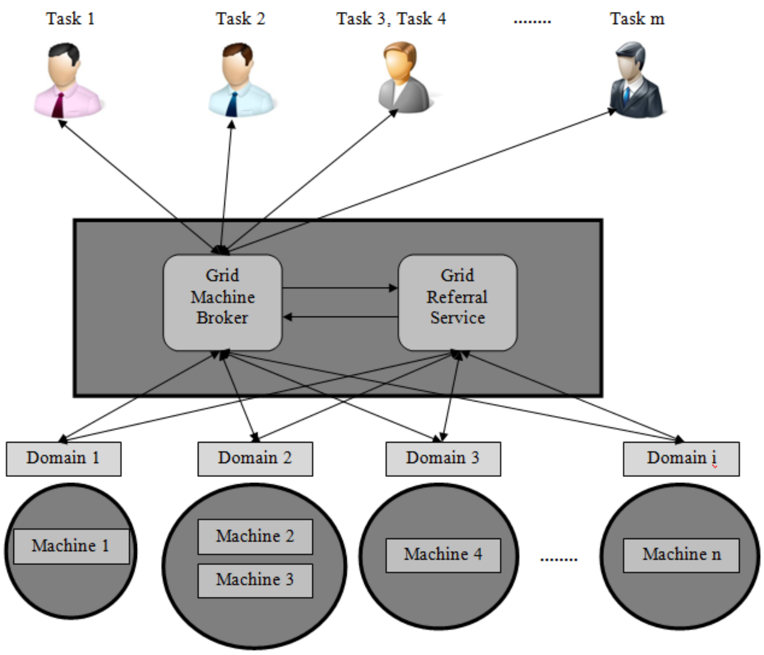

Figure 4.1: Scheduling Model

Task	**TASK_MI**	**TASK_SIZE**
T_1	100	50
T_2	25	75

Table 4.1: Grid User Task Information

4.6 Architecture of GMB

The architecture of the GMB is shown in Figure 4.2. The grid user submits the task(s) in different specifications like $TASK\_IDs$, $TASK\_MI$ (million instruction), $TASK\_SIZE$ (in megabits), $TASK\_MODE$ (immediate or batch), $TASK\_POLICY$ (preemptive or non-preemptive), $TASK\_BUDGET$, $TASK\_DEADLINE$, $TASK\_LIMIT$ and $TASK\_CATEGORY$ (high QoS or low QoS) to the GRB. After getting the details of user task(s), GMB gets the available machine information from the GRS. GRS may have different machine specifications like $MACHINE\_ID$, $MACHINE\_MIPS$ (millions instructions per second), $MACHINE\_MBPS$ (mega bits per second), $MACHINE\_PROCESSOR$, $MACHINE\_OS$, $MACHINE\_MEMORY$, $MACHINE\_COST$, $MACHINE\_SYSTEM$ and $MACHINE\_INDEX$. The above specification may vary with respect to the types of grids. Here, we have presented a general specification.

GMB starts mapping the tasks and the machines according to the specifications. It calculates the EET of a task as shown in Equation (4.1). Let us consider an example with two tasks and two machines as shown in Table 4.1 and Table 4.2 respectively. The calculated EET of the tasks using Equation (4.1) are shown in Table 4.3.

$$EET = \frac{TASK\_MI}{MACHINE\_MIPS} + \frac{TASK\_SIZE}{MACHINE\_MBPS} \qquad (4.1)$$

Machine	MACHINE_MIPS	MACHINE_MBPS
M_1	60	20
M_2	40	55

Table 4.2: Grid Machine Specification

Task / Machine	R_1	R_2
T_1	4.17	3.41
T_2	4.17	1.99

Table 4.3: EET of The Tasks on Each Machine

GMB also checks the deadline of the task as shown in Equation 4.2. It schedules the task to the machine which gives the result on or before the deadline.

$$TASK\_DEADLINE \geq EET + READYTIME \tag{4.2}$$

GRB also calculates the cost of the computation. The cost of the computation must be less than or equal to the user specified budget. The budget can be calculated as shown in Equation 4.3. Here, MACHINE_COST is calculated per second.

$$TASK\_BUDGET \geq EET \times MACHINE\_COST \tag{4.3}$$

The directory of the tasks and the machines are maintained in the GMB directory as shown in Figure 4.2.

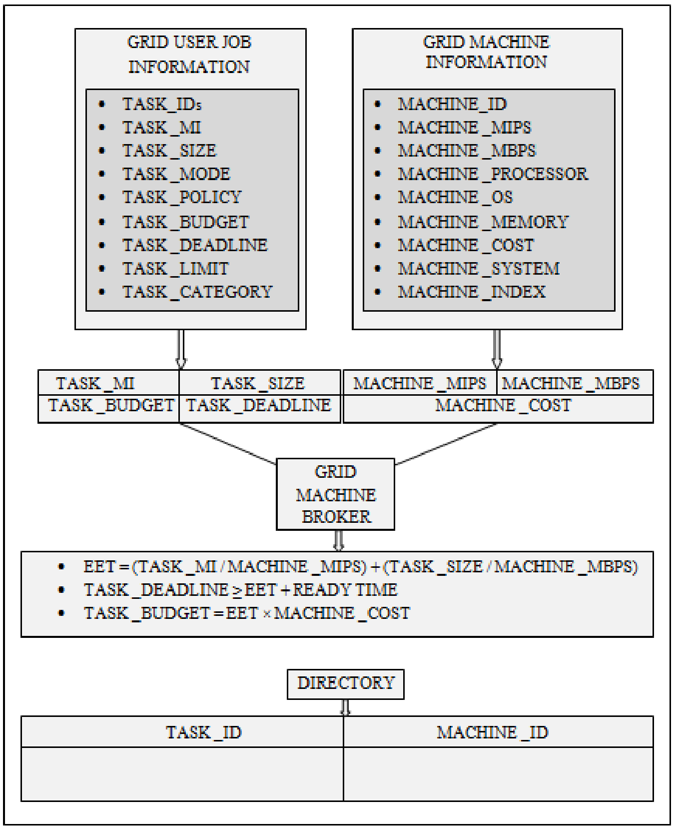

Figure 4.2: Architecture of GMB

4.7 Timeline Sequence

At first, the GMB sends a request i.e. available machine list (AML) to GRS. The GRS acknowledges by issuing AML. Then, the GMB sends the task machine lists (TMLs) to each individual domain. This list contains the mapping between the tasks and the machines. It also gives information about the machines under different domain. The domain assigns the task to the machine according to the TMLs. Finally, the results are returned back to the GMB.

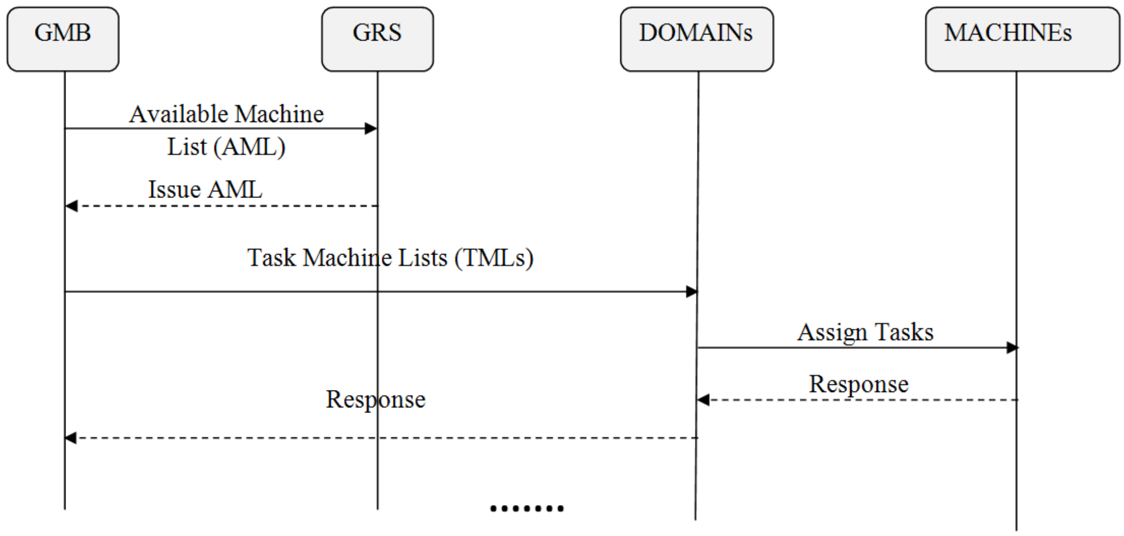

Figure 4.3: Timeline Sequence

4.8 A Research Model of Grid

A research model is described in Figure 4.4. It contains nine blocks. The scheduling algorithms are based on these nine blocks. In this thesis, we have considered computational grid, dynamic, batch, independent, preemptive and non-preemptive, all types of matrices, high QoS and low QoS, a task without duplication and all performance matrices from first to ninth block respectively.

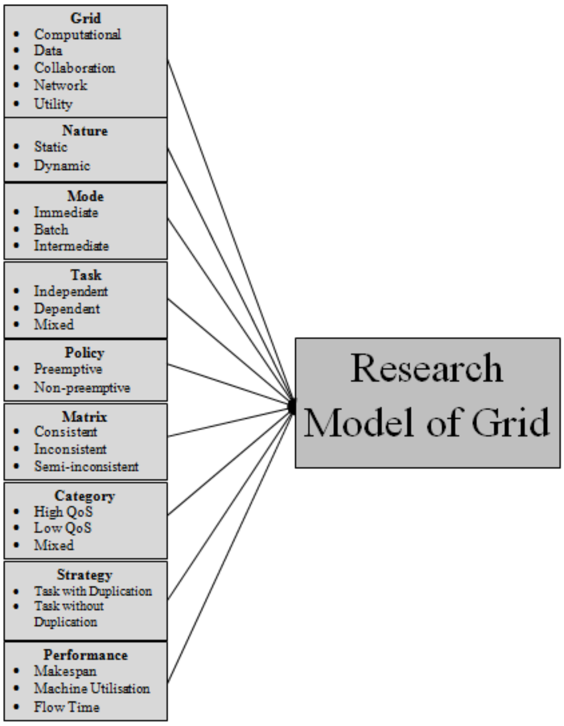

Figure 4.4: A Research Model of Grid

4.9 A Semi-Interquartile Min-Min Max-Min (SIM^2) Approach for Grid Task Scheduling

4.9.1 Heuristic Description

In this section, we present a semi-interquartile min-min max-min (SIM^2) task scheduling heuristic. At the first step, the meta-tasks are sorted in ascending order of the execution time. From the second step to the last step, all the steps are repeated until no meta-tasks are present in the TQ. In the third and fourth step, the meta-tasks are assigned to all the machines to calculate the completion time of each task in each individual processor. Completion time can be calculated using the Equation 3.2. It is shown in the fifth step.

In the step eight and nine, MCT of each meta-tasks are determined. This step gives a one dimensional array. Then, we calculate the difference between two consecutive meta-tasks MCT in step eleventh and store it in a DQ. Semi-interquartile range and Interquartile range are calculated in step twelve, using a formula shown in Equation 4.4 and 4.5 respectively.

$$Interquartilerange = Q_3 - Q_1 \tag{4.4}$$

$$Semi - Interquartilerange = \frac{Q_3 - Q_1}{2} \tag{4.5}$$

where Q_1 = First Quartile, Q_3 = Third Quartile

To calculate interquartile range, we need to find the median. Then, we divide the one dimensional array into two halves using the median. First quartile value is the median of the lower half of the array. Similarly, third quartile is the median of the upper half of the array. In step thirteen, it finds an element which is greater than the calculated semi-interquartile range and store it in location l. If no element is found, then it returns l value as null. If position l is null or greater than equal to the half of the total number of tasks i.e. $\frac{m}{2}$ then it selects max-min strategy in the first iteration. Otherwise, it selects min-min strategy. It is shown in the step fourteen to seventeen. Finally, it deletes the executed meta-task from TQ and updates the TQ in step eighteen. Then, second iteration starts to schedule another task. After all iterations are over, we calculate makespan and AMU. It is shown in the last step.

This heuristics can be mathematically expressed as follows:

Let us consider the EET matrix shown in Equation (2.1). The SIM^2 heuristic chooses the MCT of each task as shown in Equation (4.6). Then, it sorts the task to calculate the semi-interquartile range as shown in Equation (4.7). After the tasks are sorted in ascending order, it calculates the difference between two consecutive tasks as shown in (4.8). Then, it calculates first and third quartile from the one dimensional array. Finally, it calculates the semi-interquartile range.

$$\begin{pmatrix} E_{1,\alpha} \\ E_{2,\beta} \\ E_{3,\gamma} \\ \ldots \\ E_{m-1,v} \\ E_{m,v} \end{pmatrix} \tag{4.6}$$

$$sort(E_{1,\alpha}, E_{2,\beta}, E_{3,\gamma}, ..., E_{m,v}) \tag{4.7}$$

$$DQ \rightarrow E_{2,\beta} - E_{1,\alpha}, E_{3,\gamma} - E_{2,\beta}, ..., E_{m,v} - E_{m-1,v} \tag{4.8}$$

4.9.2 Heuristic

Algorithm 1 shows the semi-interquartile min-min max-min heuristic.

Algorithm 1 - Semi-Interquartile Min-Min Max-Min Heuristic

Sort the meta-tasks in ascending order of their execution time.
while $TQ\ \ != NULL$
    for all meta-tasks T_i in TQ
        for all machines M_j do
            $C_{i,j} = E_{i,j} + R_j$
        end for
    end for
    for all meta-tasks T_i in TQ
        Find minimum $C_{i,j}$ and machine M_j that holds it.
    end for
    Calculate difference between two consecutive minimum $C_{i,j}$ and Store in DQ.
    Calculate semi-interquartile range.
    Find an element e in DQ semi-interquartile range and Store the location l.
    if $l = (\frac{m}{2})$ or $l = NULL$
    then assign meta-task T_m to machine M_k that holds minimum $C_{m,k}$.
    else assign meta-task T_1 to machine M_k that holds minimum $C_{1,k}$.
    end if
    Delete the meta-task, update TQ.
end while
Calculate Makespan and AMU.

4.9.3 Illustration

Figure 4.5 shows an illustration of SIM^2 heuristic. In Figure 4.5, first example calculates the interquartile range $(IQ) = 46.5$. So, semi-interquartile range (SI) is $\frac{IQ}{2} = 23.25$. Then, it finds a location l in DQ where the SI value is greater than or equal to the value present in DQ. Here, the location l is at position number 4 because the difference between the location 5 and 4 is more than the SI value. As the location 4 is greater than equal to $\frac{5}{2}$, max-min heuristic is applied for the first iteration. Like this, in the second example, as it is less than $\frac{5}{2}$, min-min algorithm is applied for the first iteration.

Position (P):	1	2	3	4	5
CT_{ij}:	3	11	21	41 ⇧	66
				$4 \geq \frac{m}{2} = 2.5$	
IQ = 46.5, SI = 23.25, m = 5					

Position (P):	1	2	3	4	5
CT_{ij}:	1 ⇧	55	75	95	120
	$1 < \frac{m}{2} = 2.5$				
IQ = 79.5, SI = 39.75, m = 5					

Figure 4.5: Illustration of SIM^2 Heuristic

4.10 A Three-Stage Approach for Grid Task Scheduling

4.10.1 Heuristic Description

In this section, we present a three-stage approach (TSA) task scheduling heuristic. Here, three stages are used to schedule tasks. The first stage is used to find the workload of a machine. It is calculated using an average formula. Threshold and priority assignment is done in the second stage. Task allocation is started in the third stage. β and α are two scheduling metrics used in our approach. β is a matrix used in first stage. After applying threshold, α matrix is formed.

4.10.2 Heuristic

Algorithm 2 shows the semi-interquartile min-min max-min heuristic.

Algorithm 2 - Three-Stage Approach for Grid Task Scheduling

for all machines M_j
    $Avg_j = \sum_{i=1}^{m} \frac{\beta_{ij}}{m}$
end for
for all machines M_j
    for all tasks T_i
        if $\beta_{ij} > Avg_j$
        then $\alpha_{ij} = \beta_{ij}$
        else $\alpha_{ij} = 0$
        end if
    end for
end for
for all tasks T_i
    for all machines M_j
        if $\alpha_{ij} = 0$
        else $count_i = count_i + 1$
        end if
    end for
end for
Sort the tasks in descending order of their count and place it in TQ.
Repeat
    if two or more tasks having a same count value in TQ
    then Calculate SV.
        if two or more tasks having a same SV value
        then Ties are broken randomly.
        else Re-order the tasks
        end if
        Place tasks into a TEQ.
            Repeat
                for each task T_i find the optimal ECT machines M_j
                    Assign the task T_i to the machines M_j.
                    $R_j = R_j + \beta_{ij}$
                    Delete the task T_i from TQ and TEQ.
                end for
            Until the TEQ is empty.
    else find the optimal ECT machine M_j for task T_i
        Assign the task T_i to the machines M_j.
        $r_j = r_j + \beta_{Ij}$
        Delete the task T_i from TQ.
    end if
Until the TQ is empty.

4.10.3 Illustration

Let us consider an example to see how TSA approach works. In this example, we have considered 20 tasks (T_1, T_2, . . . , T_{20}) and 10 machines (M_1, M_2, . . . , M_{10}). Table 4.4 shows the ET of tasks on different machines. All values in Table 4.4 in seconds. Our approach is a three-stage approach. First, we calculate the average of all tasks on each machine. It can be calculated using a formula shown in Equation 4.9. Table 4.5 shows the Avg_j.

$$Avg_j = \sum_{i=1}^{m} \frac{\beta_{ij}}{m} \tag{4.9}$$

where β_{ij} = Task T_i on machine M_j

Avg_j = Average on machine M_j

Table 4.4: Execution Time of Tasks

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
T_1	58	40	35	82	51	85	74	55	12	74
T_2	54	45	15	43	89	56	59	63	49	16
T_3	87	36	59	89	59	93	24	13	86	87
T_4	26	77	26	39	15	70	67	62	88	94
T_5	32	63	14	77	20	58	28	36	27	99
T_6	12	77	76	40	41	82	63	25	21	86
T_7	94	92	24	81	75	88	66	49	57	79
T_8	65	98	44	76	83	99	73	19	64	51
T_9	48	19	69	38	79	20	89	12	42	17
T_{10}	64	14	36	21	32	87	98	20	30	40
T_{11}	55	70	74	79	53	61	77	14	95	13
T_{12}	65	49	39	95	29	94	58	19	78	23
T_{13}	54	53	69	33	11	53	93	24	10	94
T_{14}	72	53	71	67	13	48	58	64	14	30
T_{15}	52	86	44	44	68	80	31	28	16	29
T_{16}	90	48	41	84	50	23	12	54	62	33
T_{17}	22	86	33	19	50	87	70	57	65	94
T_{18}	10	67	42	16	49	63	50	25	65	65
T_{19}	11	74	27	14	58	85	54	94	32	42
T_{20}	26	52	19	18	99	25	85	21	44	73

Table 4.5: Average of Tasks

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
49.85	59.95	42.85	52.75	51.2	67.85	61.45	37.7	47.85	56.95

Second, we assign the priority among the tasks. Avg_j is used as a threshold to determine priority. If task T_i on machine M_j is more than the threshold (Avg_j) then it is assigned to α_{ij}. Otherwise, it is not assigned. "X" sign in Table 4.6 indicates that the corresponding task-machine pair is below the threshold value. It shows which machine contains heavy loaded tasks. Heavily loaded tasks are scheduled first in order to get a better Makespan. Table 4.6 shows the ET of tasks after the threshold is applied. For M_1, Avg_j is 49.85. Task T_1, T_2 and T_3 having 58,54 and 87 ET respectively. As these values are more than the threshold value, it is assigned to α_{ij}. Task T_4, T_5 and T_6 having 26, 32 and 12 ET respectively. But, these values are below the threshold. So, it is not assigned to α_{ij}.

For T_1, machine M_1, M_4, M_6, M_7, M_8 and M_{10} are satisfying the threshold criteria. So, T_1 has a priority (or count) 6. Alternatively, Only 4 machines are below the threshold. Similarly, T_2 has a priority 4. It indicates T_1 is less number of high speed machines than T_2. So, T_1 is processed before T_2. RQ is used to maintain the task sequence in descending order of their priority. RQ is scanned from left to right and one by one until priority is changed. For example, Task T_3 and T_{11} are having same priority i.e. 7. So, they are processed to repeat a block at the same time. It may happen that two or more tasks are assigned to same priority. In order to break the tie, we use sufferage value. Again, two or more tasks contain a same sufferage value. Finally, ties are broken randomly. In our example, T_1, T_4 and T_{17} have priority 6. SV of these tasks are 28, 11 and 3 respectively. So, Sequence order is T_1, T_4, and T_{17}. We use TQ to store the sequence temporarily.

Table 4.6: Execution Time of Tasks After Threshold

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	Count
T_1	58	X	X	82	X	85	74	55	X	74	6
T_2	54	X	X	X	89	X	X	63	49	X	4
T_3	87	X	59	89	59	93	X	X	86	87	7
T_4	X	77	X	X	X	70	67	62	88	94	6
T_5	X	63	X	77	X	X	X	X	X	99	3
T_6	X	77	76	X	X	82	63	X	X	86	5
T_7	94	92	X	81	75	88	66	49	57	79	9
T_8	65	98	44	76	83	99	73	X	64	X	8
T_9	X	X	69	X	79	X	89	X	X	X	3
T_{10}	64	X	X	X	X	87	98	X	X	X	3
T_{11}	55	70	74	79	53	X	77	X	95	X	7
T_{12}	65	X	X	95	X	94	X	X	78	X	4
T_{13}	54	X	69	X	X	X	93	X	X	94	4
T_{14}	72	X	71	67	X	X	X	64	X	X	4
T_{15}	52	86	44	X	68	80	X	X	X	X	5
T_{16}	90	X	X	84	X	X	X	54	62	X	4
T_{17}	X	86	X	X	X	87	70	57	65	94	6
T_{18}	X	67	X	X	X	X	X	X	65	65	3
T_{19}	X	74	X	X	58	85	X	94	X	X	4
T_{20}	X	X	X	X	99	X	85	X	X	73	3

4.11 RRTS: A Task Scheduling Algorithm to Minimize Makespan in Grid Environment

4.11.1 Heuristic Description

In this section, we present a round robin task scheduling to minimize makespan in Grid Environment. In our heuristic, tasks are present in the TQ and then sorted according to the fastest processors execution time. Dynamic time slice (DTS) can be calculated using a formula shown in equation 4.10. DTS is assigned to the tasks present in the task queue. Machines are assigned to the tasks based on the concept of round robin. Tasks can be switched between machines to minimize the completion time. Fastest processor remains 100 percent busy in our approach.

$$DTS = \frac{MaximumExecutionTime - MinimumExecutionTime}{TotalNumberoftasks}. \tag{4.10}$$

4.11.2 Heuristic

Algorithm 3 shows the semi-interquartile min-min max-min heuristic.

Algorithm 3 - **RRTS: A Task Scheduling Algorithm to Minimize Makespan in Grid Environment**

Select the machine M which takes less execution time for all tasks.
Sort the tasks in ascending order of their execution time. (Rest machines tasks are sorted accordingly)
Calculate dynamic time slice (DTS) = $\frac{MaximumExecutionTime - MinimumExecutionTime}{TotalNumberoftasks}$.
while $TQ\ \ != NULL$
  for i = 0 to m
    i = i mod m
    j = i mod n
    Assign TQ_i to the machine M_j
    Assign DTS to task TQ_i
    $TQ_i \rightarrow$ DTS
    $RET = ET[TQ_i] - DTS$
    if $RET == 0$
      Task TQ_i has successfully executed.
      swap();
    else if $RET > 0$
      Pre-empt the task and re-schedule it to end of the TQ.
      Update the rest machines RET.
    else if $RET < 0$
      Task has successfully executed before DTS expires.
      swap ();
    end if
  end for
  Update TQ and m.
end while

swap()

if (TQ == NULL && M == NULL)
  Pre-empt the task from machine which takes less ET after M and re-schedule it ro M.
else
  Return 0;
end if

4.11.3 Illustration

Let us consider a problem having four tasks T_0, T_1, T_2 and T_3 and two machines M_0 and M_1. It shows that m = 4 and Y = 2. Table 4.7 shows the execution time of the tasks. The procedure is shown in the following steps.

Table 4.7: Execution Time of Sorted Tasks

	M_0	M_1
T_3	1	12
T_1	2	13
T_0	3	10
T_2	5	15

1. Select the machine M which takes less ET for all tasks i.e. M_0.
2. Sort the tasks in ascending order of their ET.
3. Calculate DTS. DTS = (5 - 1) / 4 = 1. So, DTS is 1 for all the tasks.
4. TQ contains T_3, T_1, T_0 and T_2 respectively.
5. Initially, i value is 0.
6. New value of $i = i \ mod \ m = 0$.
7. Similarly, $j = i \ mod \ n = 0$.
8. T_3 is assigned to M_0.
9. Assign $DTS = 1$ to task T_3.
10. T_3 has $ET = 1$.
11. RET = 1 - 1 = 0.
12. The condition for $RET = 0$ is satisfied.
13. Task 3 has successfully executed.
14. Call swap function. As TQ not equal to $NULL$, it returns 0. Go to Step 5.
15. Now, i value is 1.

16. New value of $i = i \ mod \ m = 1$.

17. Similarly, $j = i \ mod \ n = 1$.

18. T_1 is assigned to M_1.

19. Assign $DTS = 1$ to task T_1.

20. T_1 has $ET = 13$.

21. RET = 13 - 1 = 12.

22. The condition for $RET = 0$ is not satisfied.

23. The condition for $RET > 0$ is satisfied.

24. Pre-empt the task and reschedule it to end of the TQ.

25. Update the rest machines RET. Go to Step 5. Table 4.8 shows this scenario.

Table 4.8: Execution Time of Tasks After Second Iteration

	M_0	M_1
T_1	1.85	12
T_0	3	10
T_2	5	15

4.12 Summary

In this chapter, we have proposed three batch mode heuristics: SIM^2, TSA and RRTS. These three methods are mainly used for scheduling the tasks in efficient manner. SIM^2 uses an interquartilerange concept to schedule the tasks, TSA uses thresold value concept is used and in RRTS a Round Robin concept is used to schedule the tasks efficiently. We have described the three heuristics by considering three illustrations respectively.

Chapter 5

Fault Tolerance Scheduling Heuristics for Independent Tasks in Computational Grids

5.1 Introduction

The grid failures are considered from two perspectives. First, a machine is completely failing to execute the tasks that were assigned to it. It is called a permanent fault. In this case, the task has to be assigned to the second least completion time machine or it has to be assigned in the upcoming iteration. It may be possible that the second least completion time has failed. Then, it is assigned to third least completion time machine and so on. Let us consider a task T_i that is assigned to a machine M_i.

$$T_i \rightarrow M_i$$

But, the machine M_i is faulty. So, it is not possible to map the task T_i with the machine M_i.

$$T_i \nrightarrow M_i$$

Then, the task T_i has to be assigned to the next least completion time machine M_j and so on.

$$T_i \rightarrow M_j$$
$$\text{where } ECT(M_i) < ECT(M_j)$$

Second, a machine is partially failed to execute the tasks that were assigned to it. It is called a transient fault. In this case, the task has to be assigned to the next least completion time machine until the machine available again. IRCTC website is an example of this case. The website is down in between 11:30pm to 00:30am for maintenance purpose.

5.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

The problem definition is presented in Section 5.2. In Section 5.3, fault system model is discussed. The timeline sequence for fault tolerant scheduling is presented in Section 5.4. The proposed heuristics i.e. FT-MET, FT-MCT, FT-Min-Min, FT-Max-Min are presented in Section 5.5, Section 5.6, Section 5.7 and Section 5.8 respectively.

5.3 Problem Definition

In this chapter, we focus on the problem of scheduling m tasks on n resources in a faulty environment. We have considered only the permanent fault. If a fault occurs, then the task has to be assigned in the upcoming iteration. The aim is to minimizing makespan and maximizing the machine utilisation.

5.4 Fault System Model

The fault system model considers two types of failure: machine and network link. In machine failure, the machine is not able to complete any task. But, in network failure, the task is not able to reach in the machine. Until unless the GMB get back the result from the machine, it is impossible to predict the failure i.e. machine or network link [65]. In this theis, we have considered only the machine failure.

The following methods are used to detect and prevent the fault:

5.4.1 Round Trip Time

The round trip time (RTT) is the sum of the time to send the task to a machine and acknowledge for that task. Equation 5.1 shows the RTT for task i on machine k present in domain j. Its equivalent expression shown in Equation 5.2. If GMB does not get back result within ζ of RTT than it assumes that the machine is faulty. Here, ζ varies from 1 to 2. It is shown in Equation 5.3. In this thesis, we have considered the RTT method to detect the fault.

$$RTT_{T_i \to DN_j, M_k} = D_{GMB-DN_j} + D_{DN_j} + D_{DN_j-M_k} + D_{M_k} + D_{E_{M_{T_i}}} + D_{M_k-DN_j} + D_{DN_j} + D_{DN_j-GMB} \quad (5.1)$$

$$= 2 \times (D_{GMB-DN_j} + D_{DN_j} + D_{DN_j-M_k}) + D_{M_k} + D_{E_{M_{T_i}}} \quad (5.2)$$

$RTT_{T_i \to DN_j, M_k}$ = Round trip time of task T_i on machine M_k present on domain j

D_{GRB-DN_j} = Communication delay between GMB and DN_j

D_{DN_j} = Delay on domain j including queuing delay

$D_{DN_j-M_k}$ = Communication delay between domain j and machine k

D_{M_k} = Delay on machine k including queuing delay

$D_{E_{M_{T_i}}}$ = Delay in execution time of task T_i using scheduling strategy S

$D_{M_k-DN_j}$ = Communication delay between machine k and domain j

D_{DN_j} = Delay on domain j including queuing delay

D_{DN_j-GRB} = Communication delay between DN_j and GMB

$$\begin{cases} RTT_{T_i \to DN_j, M_k} \leq \zeta \times 2 \times (D_{GMB-DN_j} + D_{DN_j} + D_{DN_j - M_k}) + D_{M_k} + D_{E_{M_{T_i}}} & M_k \; is \; not \; faulty \\ RTT_{T_i \to DN_j, M_k} > \zeta \times 2 \times (D_{GMB-DN_j} + D_{DN_j} + D_{DN_j - M_k}) + D_{M_k} + D_{E_{M_{T_i}}} & M_k \; is \; faulty \end{cases} \tag{5.3}$$

ζ = varies from 1 to 2 (depends on the types of grid)

5.4.2 Checkpointing

Checkpointing is a fault tolerance technique. It periodically saves the results on a permanent storage. If a failure happens, it goes back to the previous checkpoint state. In this thesis, we have considered the checkpointing method to prevent or recover the fault.

5.5 Timeline Sequence for Fault Tolerance Scheduling

At first, the GMB sends a request i.e. available machine list (AML) to GRS. The GRS acknowledges by issuing AML. Then, the GMB sends the task machine lists (TMLs) to each individual domain. This list contains the mapping between the tasks and the machines. It also gives information about the machines under different domain. The domain assigns the task to the machine if and only if the machine is not failed. It means the task is approved for computation. The domain acknowledges the GMB in one of the three states: approved, not approved or no response. A task is not approved because of QoS violation, requirement violation or overload machine. If there is a network failure then it is in no response state. The GMB sends the updated AML to the GRS. The GRS acknowledges to the GMB i.e. AML updated.

The GMB again sends the updated TMLs (excluding approved tasks) to the domains and the same steps are followed.

The machine sends task status list (TSLs) to the respective domains. It is in one of the states: task complete or task incomplete. The domains send the list to the GMB. Finally, the GMB acknowledges for TSLs to the respective domains. The process is repeated until the GMB does not contain any task.

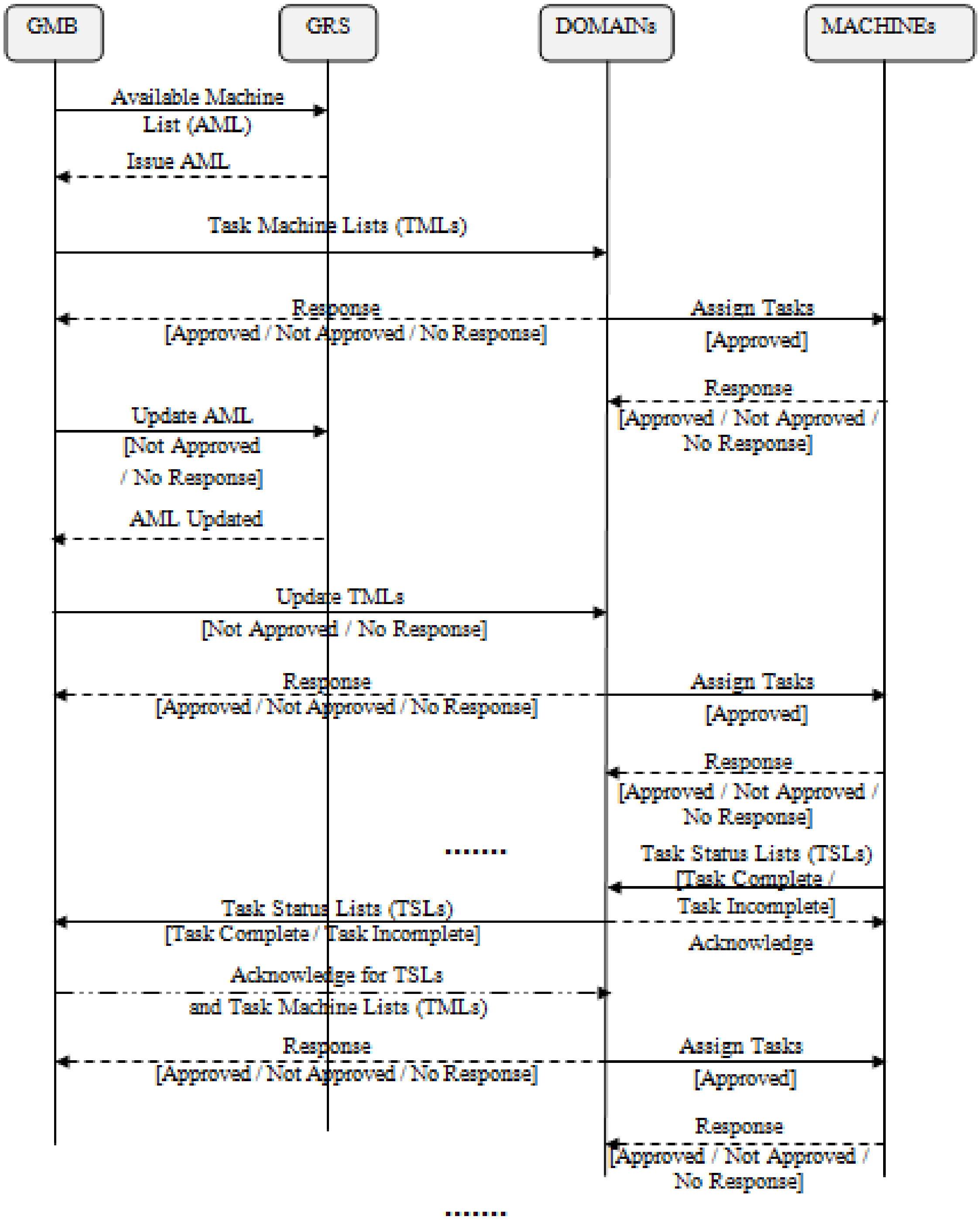

Figure 5.1: Timeline Sequence for Fault Tolerance Scheduling

5.6 Fault Tolerant - Minimum Execution Time Heuristic

5.6.1 Heuristic Description

The heuristic is divided into two phases: matching and scheduling. The matching phase is similar to the existing MET heuristic. Lines 1 to 3 in Algorithm 4 show the matching phase. But, in scheduling phase, the GRB gets the current status of the machine from GRS. If the machine is faulty, it finds the next least execution time machine. Then, it again checks the status of the machine. If the machine is not faulty, it assigns the task to the machine. Line 5 to 10 in Algorithm 4 show the scheduling phase.

5.6.2 Heuristic

Algorithm 4 shows the fault tolerant - minimum execution time heuristic.

Algorithm 4 - **Fault Tolerant - Minimum Execution Time Heuristic**

1: **for** task T_i
2:     **for** all machines M_j
3:         Find minimum $E_{i,j}$ and machine M_j that holds it.
4:         Set $k = 1$.
5:         Find the status of M_j from GRS.
6:         **if** $(M_j == Faulty)$
7:             Find $(k+1)$ minimum $E_{i,j}$ for T_i and machine M_j that holds it.
8:             Go to Step 5.
9:         **else** Assign task T_i to machine M_j
10:        **end if**
11:    **end for**
12: **end for**

5.6.3 Illustration

Let us consider a problem consisting three machines M_1, M_2 and M_3 and four tasks T_1, T_2, T_3 and T_4. Table 5.1 shows the EET matrix for 4 tasks and 3 machines.

For task T_1, the least execution time machine is M_3. So, it is assigned to machine M_3. Like task T_1, the least execution time for task T_2, task T_3 and task T_4 is machine M_1, machine M_2, machine M_1 respectively. So, the overall makespan is 63.

Assume that, the machine M_1 is failed due to some unavoidable circumstance. So, the task T_2 and the task T_4 are not computed successfully. The overall makespan is reduced to 32.

In our proposed heuristic, if the machine M_1 is failed due to some unavoidable circumstance, then the task T_2 and the task T_4 are assigned to machine M_3 and M_2 respectively. The overall makespan is 125.

Table 5.1: EET Matrix for 4 Tasks and 3 Machines

Task / Machine	M_1	M_2	M_3
T_1	120	75	32
T_2	40	110	93
T_3	71	24	49
T_4	23	34	47

5.7 Fault Tolerant - Minimum Completion Time Heuristic

5.7.1 Heuristic Description

The heuristic is divided into two phases: matching and scheduling. The matching phase is similar to the existing MCT heuristic. Lines 1 to 7 in Algorithm 5 show the matching phase. But, in scheduling phase, the GRB gets the current status of the machine from GRS. If the machine is faulty, it finds the next least completion time machine. Then, it again checks the status of the machine. If the machine is not faulty, it assigns the task to the machine. Line 9 to 14 in Algorithm 5 show the scheduling phase.

5.7.2 Heuristic

Algorithm 5 shows the fault tolerant - minimum completion time heuristic.

Algorithm 5 - **Fault Tolerant - Minimum Completion Time Heuristic**

1: **for** task T_i
2:     **for** all machines M_j
3:         $C_{i,j} = E_{i,j} + R_j$
4:     **end for**
5: **end for**
6: **for** task T_i
7:     Find minimum $C_{i,j}$ and machine M_j that holds it.
8:     Set $k = 1$.
9:     Find the status of M_j from GRS.
10:     **if** ($M_j == Faulty$)
11:         Find $(k + 1)$ minimum $C_{i,j}$ for T_i and machine M_j that holds it.
12:         Go to Step 9.
13:     **else** Assign task T_i to machine M_j
14:     **end if**
15: **end for**

5.7.3 Illustration

Let us consider a problem consisting three machines M_1, M_2 and M_3 and four tasks T_1, T_2, T_3 and T_4. Table 5.1 shows the EET matrix for 4 tasks and 3 machines.

For task T_1, the least completion time machine is M_3. So, it is assigned to machine M_3. Like task T_1, the least completion time for task T_2, task T_3 and task T_4 is machine M_1, machine M_2, machine M_2 respectively. So, the overall makespan is 58.

Assume that, the machine M_1 is failed due to some unavoidable circumstance. So, the task T_2 is not computed successfully. The overall makespan is 58.

In our proposed heuristic, if the machine M_1 is failed due to some unavoidable circumstance, then the task T_2 is assigned to machine M_2. The overall makespan is 128.

5.8 Fault Tolerant - Min-Min Heuristic

5.8.1 Heuristic Description

The heuristic is divided into two phases: matching and scheduling. The matching phase is similar to the existing min-min heuristic. Lines 1 to 9 in Algorithm 6 show the matching phase. But, in scheduling phase, the GRB gets the current status of the machine from GRS. If the machine is faulty, it finds the next least completion time task. Then, it again checks the status of the machine. If the machine is not faulty, it assigns the task to the machine. Line 10 to 14 in Algorithm 6 show the scheduling phase.

5.8.2 Heuristic

Algorithm 6 shows the fault tolerant - min-min heuristic.

Algorithm 6 - **Fault Tolerant - Min-Min Heuristic**

1: **for** all tasks T_i in TQ
2: **for** all machines M_j
3: $C_{i,j} = E_{i,j} + R_j$
4: **end for**
5: **end for**
6: **for** all tasks T_i in TQ
7: Find minimum $C_{i,j}$ and machine M_j that holds it.
8: **end for**
9: Find the task T_h with the minimum $C_{i,j}$ and machine M_j that holds it.
10: Find the status of M_j from GRS.
11: **if** $(M_j == Faulty)$
12: Go to Step 9.
13: **else** Assign task T_h to machine M_j that gives minimum $C_{i,j}$
14: **end if**
15: Delete the task T_h from TQ and Update R_j.

5.8.3 Illustration

Let us consider a problem consisting three machines M_1, M_2 and M_3 and four tasks T_1, T_2, T_3 and T_4. Table 5.1 shows the EET matrix for 4 tasks and 3 machines.

For task T_1, the least completion time machine is M_3. Like task T_1, the least completion time for task T_2, task T_3 and task T_4 is machine M_1, machine M_2, machine M_1 respectively. But, the least completion time among all the tasks are task T_4. So, task T_4 is assigned to the machine M_1. Then, task T_3, task T_1 and task T_2 are assigned to machine M_2, machine M_3 and machine M_1 respectively. So, the overall makespan is 63.

Assume that, the machine M_1 is failed due to some unavoidable circumstance. So, the task T_4 and the task T_2 are not computed successfully. The overall makespan is 32.

In our proposed heuristic, if the machine M_1 is failed due to some unavoidable circumstance, then the task T_4 and the task T_2 are assigned to the machine M_2 and machine M_3 respectively. The overall makespan is 125.

5.9 Fault Tolerant - Max-Min Heuristic

5.9.1 Heuristic Description

The heuristic is divided into two phases: matching and scheduling. The matching phase is similar to the existing max-min heuristic. Lines 1 to 9 in Algorithm 7 show the matching phase. But, in scheduling phase, the GRB gets the current status of the machine from GRS. If the machine is faulty, it finds the next least completion time task. Then, it again checks the status of the machine. If the machine is not faulty, it assigns the task to the machine. Line 10 to 14 in Algorithm 7 show the scheduling phase.

5.9.2 Heuristic

Algorithm 7 shows the fault tolerant - max-min heuristic.

Algorithm 7 - **Fault Tolerant - Max-Min Heuristic**

```
1: for all tasks T_i in TQ
2:     for all machines M_j
3:         C_{i,j} = E_{i,j} + R_j
4:     end for
5: end for
6: for all tasks T_i in TQ
7:     Find minimum C_{i,j} and machine M_j that holds it.
8: end for
9: Find the task T_h with the maximum C_{i,j} and machine M_j that holds it.
10: Find the status of M_j from GRS.
11: if (M_j == Faulty)
12:     Go to Step 9.
13: else Assign task T_h to machine M_j that gives minimum C_{i,j}
14: end if
15: Delete the task T_h from TQ and Update R_j.
```

5.9.3 Illustration

Let us consider a problem consisting three machines M_1, M_2 and M_3 and four tasks T_1, T_2, T_3 and T_4. Table 5.1 shows the EET matrix for 4 tasks and 3 machines.

For task T_1, the least completion time machine is M_3. Like task T_1, the least completion time for task T_2, task T_3 and task T_4 is machine M_1, machine M_2, machine M_1 respectively. But, the utmost completion time among all the tasks are task T_2. So, task T_2 is assigned to the machine M_1. Then, task T_4, task T_3 and task T_1 are assigned to machine M_2, machine M_3 and machine M_3 respectively. So, the overall makespan is 81.

Assume that, the machine M_1 is failed due to some unavoidable circumstance. So, the task T_1 is not computed successfully. The overall makespan is 81.

In our proposed heuristic, if the machine M_1 is failed due to some unavoidable circumstance, then the task T_2 is assigned to the machine M_3. The overall makespan is 133.

5.10 Summary

In this chapter, we have proposed four fault tolerant batch mode heuristics: FT-MET, FT-MCT, FT-Min-Min and FT-Max-Min. We have added the concepts of fault tolerance in these heuristics to evaluate the performance of the methods when fault arises in the machine. Each heuristic is discussed with an illustration to show the fault tolerant scheme.

Chapter 6

Implementation and Results

6.1 Introduction

In this section, we present some performance evaluation strategies (or performance measures) which are used to compare the performance of existing works and our heuristics. The performance evaluation strategies include makespan, machine utilisation, completion time and idle time. But, we have considered two performance measures: makespan and machine utilisation. The implementation and results are compared based on the performance evaluation strategies. We simulated the proposed heuristics using MATLAB R2010b version 7.11.0.584.

6.2 Chapter Organisation

The organisation of the rest of the chapter and a brief outline of the sections is as follows.

The implementation details are discussed in Section 6.2. In Section 6.3, we have discussed various performance measures use to evaluate the heuristics. The results are shown in Section 6.4.

6.3 Implementation Details

6.3.1 Data Set

We have taken Braun et al. data sets (or instances) to evaluate the proposed heuristics [17]. The general form of the data sets is u_t_mmnn. Here, u indicates the uniform distribution, t indicates the types of matrices: consistent, inconsistent and semi-consistent, mm indicates the task heterogeneity and nn indicates the machine heterogeneity. The value of mm or nn is either hi or lo. So, each type of matrix contains four data sets such as hihi, hilo, lohi and lolo. Finally, we have 12 data sets. The data sets are u_c_hihi, u_c_hilo, u_c_lohi, u_c_lolo, u_i_hihi, u_i_hilo, u_i_lohi, u_i_lolo, u_s_hihi, u_s_hilo, u_s_lohi and u_s_lolo. The size of the data sets is 512 × 16, 1024 × 32 and 2048 × 64. Here, the first value indicates the number of tasks and the second value indicates the number of machines.

Apart from the above data sets, we have taken our own data sets to evaluate the performance of some heuristics. These data sets are generated using the MATLAB random function. The data sets are 50 × 5, 50 × 10, 50 × 15, 100 × 5, 100 × 10, 100 × 15, 1000 × 5, 1000 × 10, 1000 × 15, 10000 × 5, 10000 × 10 and 10000 × 15. Here, the first value indicates the number of tasks and the second value indicates the number of machines.

6.4 Performance Evaluation Strategies

6.4.1 Makespan

The makespan is the maximum completion time taken to assign all tasks to the machine. It is used to measure the throughput of the grid. It can be mathematically expressed as follows:
The makespan of the first machine using scheduling strategy S is:

$$M(S_{M_1}) = (E_{1,1} \times F_{1,1}) + (E_{2,1} \times F_{2,1}) + (E_{3,1} \times F_{3,1}) + ... + (E_{m,1} \times F_{m,1})$$

The makespan of the second machine using scheduling strategy S is:

$$M(S_{M_2}) = (E_{1,2} \times F_{1,2}) + (E_{2,2} \times F_{2,2}) + (E_{3,2} \times F_{3,2}) + ... + (E_{m,2} \times F_{m,2})$$

The makespan of the third machine using scheduling strategy S is:

$$M(S_{M_3}) = (E_{1,3} \times F_{1,3}) + (E_{2,3} \times F_{2,3}) + (E_{3,3} \times F_{3,3}) + ... + (E_{m,3} \times F_{m,3})$$

....

The makespan of the n^{th} machine using scheduling strategy S is:

$$M(S_{M_n}) = (E_{1,n} \times F_{1,n}) + (E_{2,n} \times F_{2,n}) + (E_{3,n} \times F_{3,n}) + ... + (E_{m,n} \times F_{m,n})$$

$$\text{where } F_{i,j} = \begin{Bmatrix} 1 & if\ T_i \longrightarrow M_j \\ 0 & Otherwise \end{Bmatrix}$$

The overall makespan is:

$$M(S) = max(M(S_{M_1}), M(S_{M_2}), M(S_{M_3}), ..., M(S_{M_n}))$$

(or)

$$M(S) = max(\textstyle\sum_{i=1}^{m} E_{i,1} \times F_{i,1}, \sum_{i=1}^{m} E_{i,2} \times F_{i,2}, \sum_{i=1}^{m} E_{i,3} \times F_{i,3}, ..., \sum_{i=1}^{m} E_{i,n} \times F_{i,n})$$

6.4.2 Completion Time

The completion time is the sum of the completion times of the tasks. It can be mathematically expressed as follows:

The completion time of the first task using scheduling strategy S is:

$$F(S_{T_1}) = (E_{1,1} \times F_{1,1}) + (E_{1,2} \times F_{1,2}) + (E_{1,3} \times F_{1,3}) + ... + (E_{1,n} \times F_{1,n})$$

The completion time of the second task using scheduling strategy S is:

$$F(S_{T_2}) = (E_{2,1} \times F_{2,1}) + (E_{2,2} \times F_{2,2}) + (E_{2,3} \times F_{2,3}) + ... + (E_{2,n} \times F_{2,n})$$

The completion time of the third task using scheduling strategy S is:

$$F(S_{T_3}) = (E_{3,1} \times F_{3,1}) + (E_{3,2} \times F_{3,2}) + (E_{3,3} \times F_{3,3}) + ... + (E_{3,n} \times F_{3,n})$$

....

The completion time of the m^{th} task using scheduling strategy S is:

$$F(S_{T_m}) = (E_{m,1} \times F_{m,1}) + (E_{m,2} \times F_{m,2}) + (E_{m,3} \times F_{m,3}) + ... + (E_{m,n} \times F_{m,n})$$

$$\text{where } F_{i,j} = \left\{\begin{matrix} 1 & if\ T_i \longrightarrow M_j \\ 0 & Otherwise \end{matrix}\right\}$$

The sum of completion times is:

$$F(S) = (F(S_{T_1}) + F(S_{T_2}) + F(S_{T_3}) + ... + F(S_{T_m}))$$

(or)

$$F(S) = \sum_{i=1}^{m} \sum_{j=1}^{n} E_{i,j} \times F_{i,j}$$

6.4.3 Machine Utilisation

The machine utilisation is the time that the machine is busy. It can be mathematically expressed as follows:
The machine utilisation of the first machine using scheduling strategy S is:

$$MU(S_{M_1}) = \frac{M(S_{M_1})}{M(S)}$$

The machine utilisation of the second machine using scheduling strategy S is:

$$MU(S_{M_2}) = \frac{M(S_{M_2})}{M(S)}$$

The machine utilisation of the third machine using scheduling strategy S is:

$$MU(S_{M_3}) = \frac{M(S_{M_3})}{M(S)}$$

....

The machine utilisation of the n^{th} machine using scheduling strategy S is:

$$MU(S_{M_n}) = \frac{M(S_{M_n})}{M(S)}$$

The average machine utilisation is:

$$MU(S) = \frac{MU(S_{M_1})+MU(S_{M_2})+MU(S_{M_3})+...+MU(S_{M_n})}{n}$$

The average machine utilisation (in percentage) is:

$$\%MU(S) = \frac{MU(S_{M_1})+MU(S_{M_2})+MU(S_{M_3})+...+MU(S_{M_n})}{n} \times 100$$

6.4.4 Idle Time

The idle time is the time that the machine is idle. It can be mathematically expressed as follows:

The idle time of the first machine using scheduling strategy S is:

$$I(S_{M_1}) = \frac{M(S)-M(S_{M_1})}{M(S)}$$

The idle time of the second machine using scheduling strategy S is:

$$I(S_{M_2}) = \frac{M(S)-M(S_{M_2})}{M(S)}$$

The idle time of the third machine using scheduling strategy S is:

$$I(S_{M_3}) = \frac{M(S)-M(S_{M_3})}{M(S)}$$

....

The idle time of the n^{th} machine using scheduling strategy S is:

$$I(S_{M_n}) = \frac{M(S)-M(S_{M_n})}{M(S)}$$

In general, the idle time of the machine i using scheduling strategy S is:

$$I(S_{M_i}) = \left\{ \begin{array}{cc} \frac{M(S)-M(S_{M_i})}{M(S)} & if\ M(S) \neq M(S_{M_i}) \\ 1 & Otherwise \end{array} \right\}$$

The average idle time is:

$$I(S) = \frac{I(S_{M_1})+I(S_{M_2})+I(S_{M_3})+...+I(S_{M_n})}{n}$$

The average idle time (in percentage) is:

$$\%I(S) = \frac{I(S_{M_1})+I(S_{M_2})+I(S_{M_3})+...+I(S_{M_n})}{n} \times 100$$

6.5 Results

6.5.1 Results of SIM^2 heuristic

The comparison of makespan and machine utilisation for min-min, max-min and the proposed SIM^2 heuristic are shown in Table 6.1 and Table 6.2 respectively. The graphical representation of makespan and machine utilisation are shown in Figure 6.1 and Figure 6.2 respectively. The results show that the SIM^2 heuristic is performing best amongst all. The makespan of max-min and SIM^2 heuristic are same for 50 and 100 tasks, but the overall performance of the SIM^2 heuristic is better. The machine utilisation is almost same for max-min and SIM^2 heuristic, but the overall performance of the max-min heuristic is better.

Let us consider a task processing system. We have two tasks: T_i and T_j. The tasks are processed in a heterogeneous environment. Let task T_i completes its execution before the task T_j. Obviously, the task T_i has better makespan in comparison to the task T_j. It may not indicate that the machine utilisation of the task T_i is better than the task T_j. Because, the machine utilisation is calculated from the respective makespan.

Table 6.1: Makespan Values for Min-Min, Max-Min and SIM^2 Heuristic

Instances	**Min-Min**	**Max-Min**	**SIM^2**
50 × 5	10918	**9191**	**9191**
50 × 10	9964	**9190**	**9190**
50 × 15	9762	**9190**	**9190**
100 × 5	12849	**9190**	**9190**
100 × 10	10915	**9190**	**9190**
100 × 15	10331	**9190**	**9190**
1000 × 5	47301	41523	**40199**
1000 × 10	28068	20860	**20025**
1000 × 15	21750	13918	**13333**
10000 × 5	391736	398941	**384634**
10000 × 10	199589	200235	**191539**
10000 × 15	135925	133733	**127505**

Table 6.2: Machine Utilisation Values for Min-Min, Max-Min and SIM^2 Heuristic

Instances	**Min-Min**	**Max-Min**	**SIM^2**
50 × 5	0.3399	**0.4100**	0.4050
50 × 10	0.1862	**0.2041**	0.2027
50 × 15	0.1265	**0.1356**	0.1346
100 × 5	0.4386	**0.6252**	0.6135
100 × 10	0.2572	**0.3114**	0.3057
100 × 15	0.1809	**0.2068**	0.2044
1000 × 5	0.8474	**0.9978**	0.9972
1000 × 10	0.7111	0.9953	**0.9970**
1000 × 15	0.6109	0.9941	**0.9970**
10000 × 5	0.9816	**0.9998**	0.9997
10000 × 10	0.9594	0.9995	**0.9997**
10000 × 15	0.9377	0.9993	**0.9997**

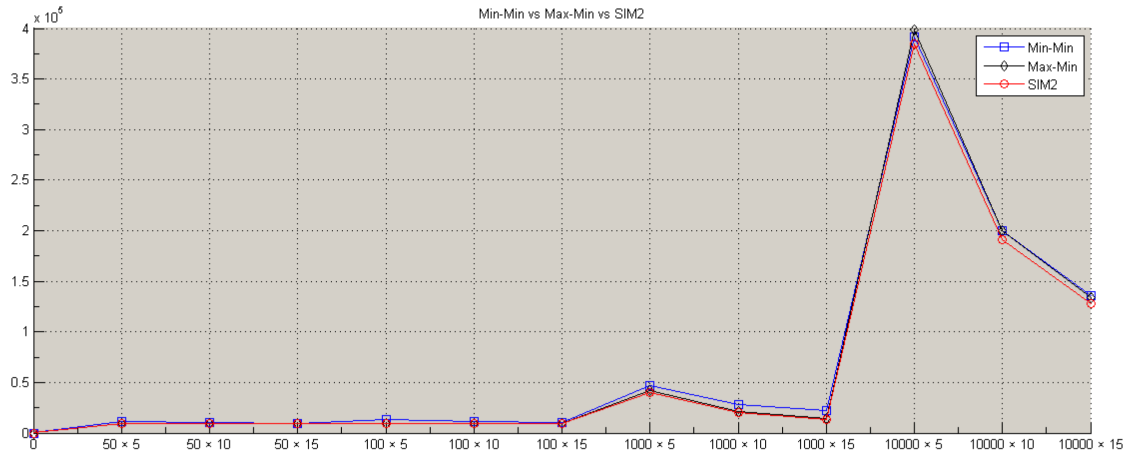

Figure 6.1: Makespan for Min-Min vs Makespan for Max-Min vs Makespan for SIM^2

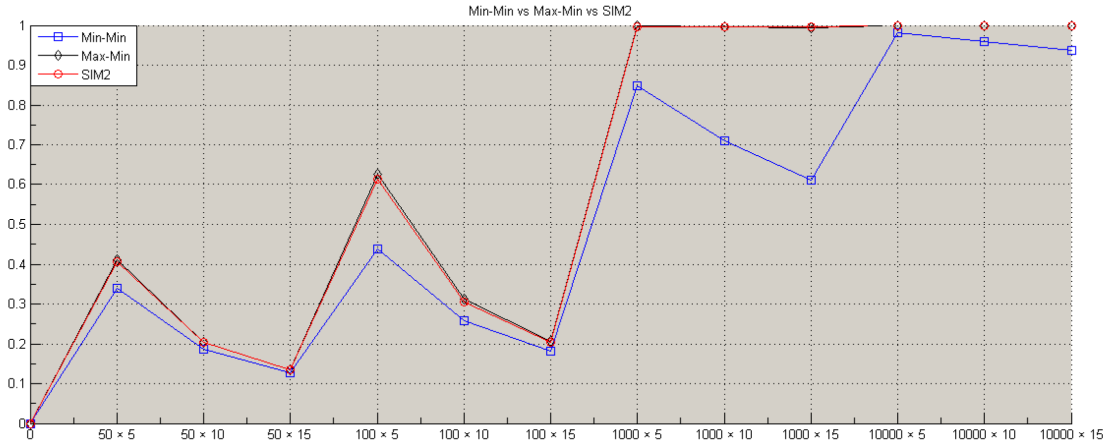

Figure 6.2: Machine Utilisation for Min-Min vs Makespan for Max-Min vs Makespan for SIM^2

6.5.2 Results of TSA heuristic

The comparison of makespan and machine utilisation for min-min, max-min and the proposed TSA heuristic are shown in Table 6.3 and Table 6.4 respectively. The graphical representation of makespan and machine utilisation are shown in Figure 6.3 and Figure 6.4 respectively. The results show that the TSA heuristic is performing best amongst all. The makespan of max-min and TSA heuristic are same for 50, 100 and 10000 tasks, but the overall performance of the TSA heuristic is better. The machine utilisation is almost same for max-min and TSA heuristic, but the overall performance of the max-min heuristic is better.

Table 6.3: Makespan Values for Min-Min, Max-Min and TSA Heuristic

Instances	**Min-Min**	**Max-Min**	**TSA**
50 × 5	12023	9831	**9765**
50 × 10	10510	**9301**	**9301**
50 × 15	10201	**9300**	**9300**
100 × 5	19045	14085	**14032**
100 × 10	12021	**9303**	**9303**
100 × 15	1356	1329	**1252**
1000 × 5	78848	81964	**78573**
1000 × 10	41502	40530	**38435**
1000 × 15	29294	26468	**24999**
10000 × 5	715161	702185	**665857**
10000 × 10	406324	365003	**341162**
10000 × 15	756937	**555303**	**555303**

Table 6.4: Machine Utilisation Values for Min-Min, Max-Min and TSA Heuristic

Instances	**Min-Min**	**Max-Min**	**TSA**
50 × 5	0.7999	0.9838	**0.9877**
50 × 10	0.4382	**0.4977**	0.4965
50 × 15	0.2855	**0.2593**	0.2589
100 × 5	0.7263	**0.9910**	0.9905
100 × 10	0.4916	**0.6416**	0.6383
100 × 15	0.7435	**0.9818**	0.9200
1000 × 5	0.9596	0.9983	**0.9984**
1000 × 10	0.8826	**0.9976**	0.9952
1000 × 15	0.8121	**0.9953**	0.9936
10000 × 5	0.8891	0.9998	**0.9999**
10000 × 10	0.7993	0.9996	**0.9997**
10000 × 15	0.3310	**0.5014**	0.4709

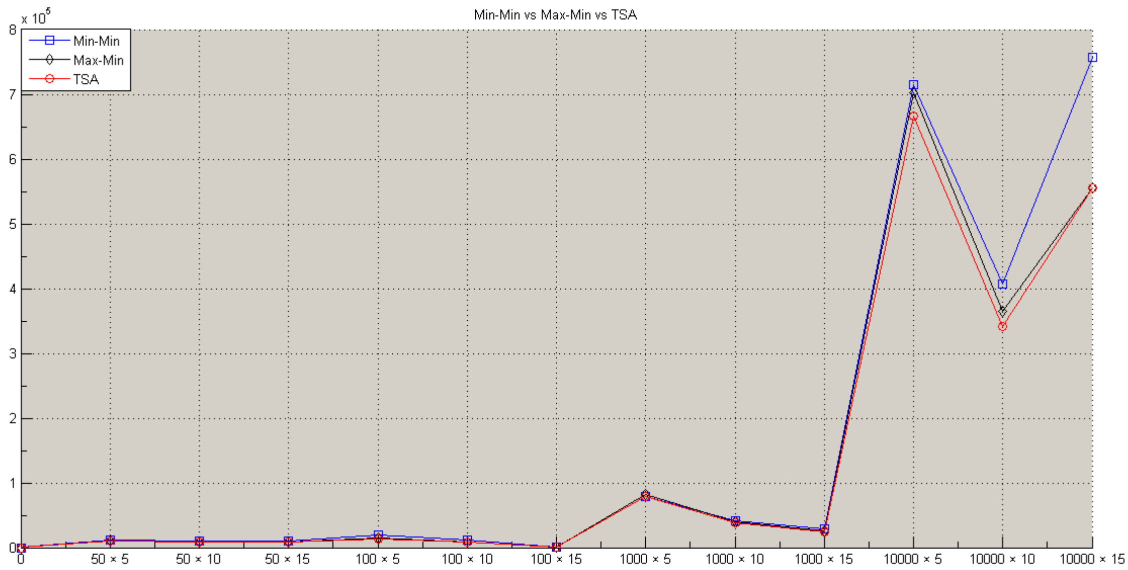

Figure 6.3: Makespan for Min-Min vs Makespan for Max-Min vs Makespan for TSA

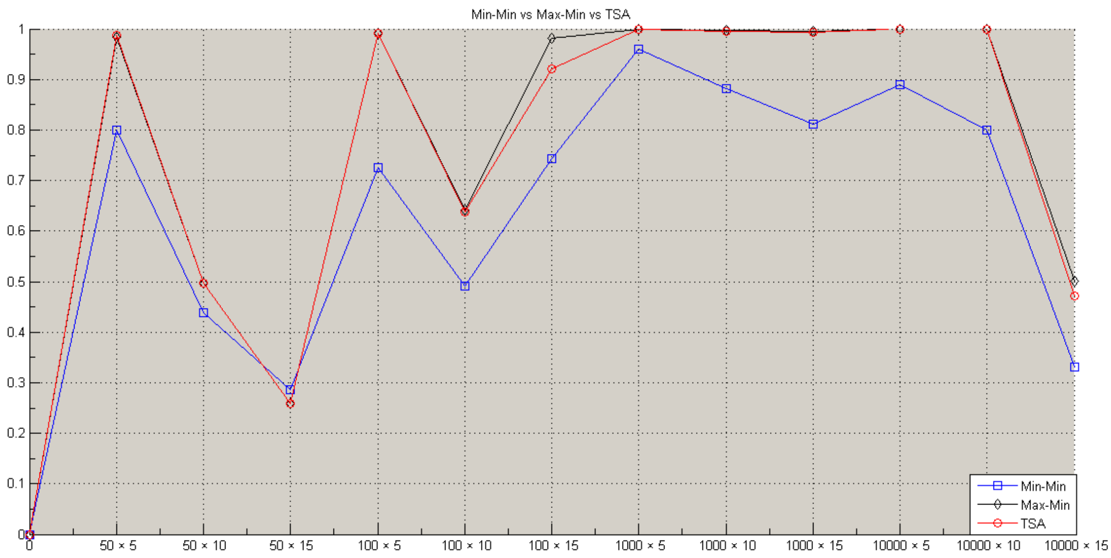

Figure 6.4: Machine Utilisation for Min-Min vs Makespan for Max-Min vs Makespan for TSA

6.5.3 Results of RRTS heuristic

The comparison of makespan for min-min, max-min and the proposed $RRTS$ heuristic are shown in Table 6.5. The graphical representation of makespan is shown in Figure 6.5. The results show that the $RRTS$ heuristic is performing best amongst all. The makespan of $RRTS$ heuristic is better than other heuristics.

Table 6.5: Makespan Values for Min-Min, Max-Min and RRTS Heuristic

Instances	**Min-Min**	**Max-Min**	**RRTS**
50 × 5	1.2023E+04	9.8310E+03	**9.7308E+03**
50 × 10	1.0510E+04	9.3010E+03	**4.6805E+03**
50 × 15	1.0201E+04	9.3000E+03	**2.5203E+03**
100 × 5	1.9045E+04	1.4085E+04	**1.3956E+04**
100 × 10	1.2021E+04	9.3030E+03	**6.0347E+03**
100 × 15	1.0331E+04	9.1900E+03	**1.9815E+03**
1000 × 5	8.3626E+04	8.5492E+04	**8.2265E+04**
1000 × 10	4.1502E+04	4.0530E+04	**3.8647E+04**
1000 × 15	2.9294E+04	2.6468E+04	**2.5214E+04**
10000 × 5	7.1516E+05	7.0218E+05	**6.6879E+05**
10000 × 10	4.0632E+05	3.6500E+05	**3.4500E+05**
10000 × 15	7.5693E+05	5.5530E+05	**2.6508E+05**

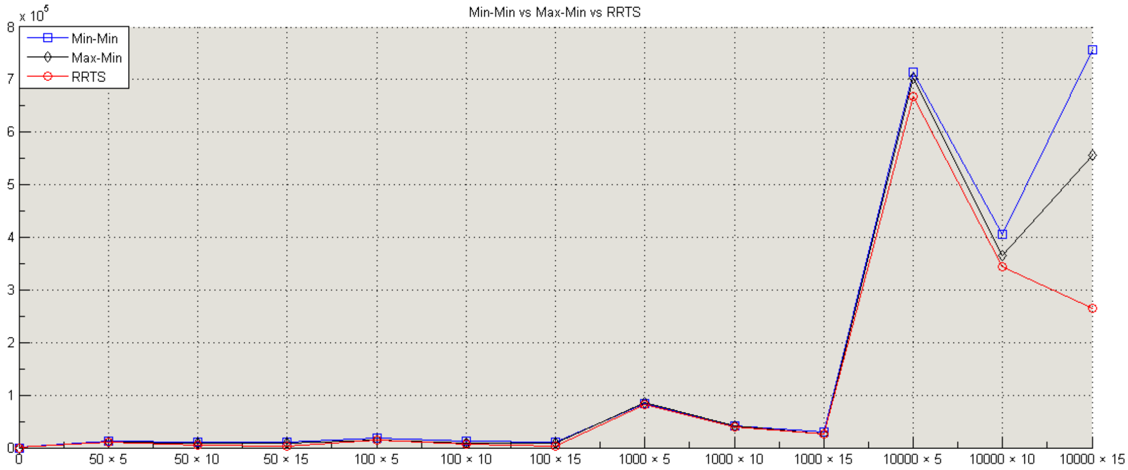

Figure 6.5: Makespan for Min-Min vs Makespan for Max-Min vs Makespan for RRTS

6.5.4 Results of MET, MCT, Min-Min and Max-Min Heuristics Without Fault Tolerance

The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 512 × 16 data sets are shown in Table 6.6 and Table 6.7 respectively. The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 1024 × 32 data sets are shown in Table 6.8 and Table 6.9 respectively. The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 2048 × 64 data sets are shown in Table 6.10 and Table 6.11 respectively.

Table 6.6: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (512 × 16 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	2.2159E+07	1.1423E+07	8.1189E+06	1.2382E+07
u_c_hilo	5.3951E+05	1.8589E+05	1.6181E+05	2.0405E+05
u_c_lohi	6.6846E+05	3.7830E+05	2.6700E+05	3.9247E+05
u_c_lolo	1.8065E+04	6.3601E+03	5.4255E+03	6.9443E+03
u_i_hihi	3.7073E+06	4.4136E+06	3.5139E+06	8.0184E+06
u_i_hilo	9.4796E+04	9.4856E+04	8.0756E+04	1.5191E+05
u_i_lohi	1.4232E+05	1.4382E+05	1.0897E+05	2.5153E+05
u_i_lolo	3.3993E+03	3.1374E+03	2.6401E+03	5.1766E+03
u_s_hihi	1.1077E+07	6.4227E+06	4.8348E+06	9.1951E+06
u_s_hilo	2.7135E+05	1.1837E+05	1.0327E+05	1.7262E+05
u_s_lohi	3.0255E+05	1.8409E+05	1.3738E+05	2.8205E+05
u_s_hilo	8.6922E+03	4.4361E+03	3.8068E+03	6.2318E+03

Table 6.7: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (512 × 16 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	1	0.7020	0.5234	0.8769
u_c_hilo	1	0.7090	0.5909	0.8519
u_c_lohi	1	0.7054	0.5347	0.8547
u_c_lolo	1	0.6980	0.5895	0.8536
u_i_hihi	0.5741	0.7077	0.4745	0.8842
u_i_hilo	0.5711	0.7016	0.5728	0.8522
u_i_lohi	0.5002	0.6961	0.5262	0.8760
u_i_lolo	0.5587	0.7202	0.5939	0.8539
u_s_hihi	0.2420	0.7119	0.4726	0.8740
u_s_hilo	0.2660	0.7339	0.5653	0.8602
u_s_lohi	0.2863	0.7103	0.5030	0.8844
u_s_hilo	0.3109	0.7093	0.5753	0.8576

Table 6.8: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (1024 × 32 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	3.8431E+07	3.1749E+07	2.0735E+07	3.2007E+07
u_c_hilo	3.6308E+06	3.1614E+06	2.1880E+06	3.2199E+06
u_c_lohi	3.2742E+03	2.8765E+03	2.0370E+03	3.1182E+03
u_c_lolo	3.7785E+02	3.2576E+02	2.2587E+02	3.2910E+02
u_i_hihi	6.7612E+06	7.4194E+06	5.9639E+06	1.3223E+07
u_i_hilo	6.7070E+05	6.7008E+05	5.5055E+05	1.2517E+06
u_i_lohi	8.5439E+02	7.5134E+02	6.2358E+02	1.3313E+03
u_i_lolo	9.1120E+01	6.9460E+01	6.3720E+01	1.2753E+02
u_s_hihi	2.4737E+07	1.7347E+07	1.3558E+07	2.3282E+07
u_s_hilo	2.2116E+06	1.7473E+06	1.3175E+06	2.2329E+06
u_s_lohi	2.1260E+03	1.6444E+03	1.3546E+03	2.2049E+03
u_s_hilo	1.7873E+02	1.8050E+02	1.2871E+02	2.2347E+02

Table 6.9: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (1024 × 32 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	1	0.6475	0.4745	0.8060
u_c_hilo	1	0.6495	0.4578	0.8007
u_c_lohi	1	0.6480	0.4540	0.8046
u_c_lolo	1	0.6448	0.4502	0.8049
u_i_hihi	0.4994	0.6176	0.4317	0.8095
u_i_hilo	0.4495	0.6428	0.4560	0.8195
u_i_lohi	0.3730	0.6058	0.4288	0.8025
u_i_lolo	0.3546	0.6419	0.4182	0.8172
u_s_hihi	0.0928	0.6704	0.3854	0.8380
u_s_hilo	0.1116	0.6471	0.3767	0.8284
u_s_lohi	0.1137	0.6696	0.3877	0.8376
u_s_hilo	0.1184	0.6317	0.3788	0.8232

Table 6.10: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (2048 × 64 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	1.6736E+07	2.7362E+07	1.8372E+07	2.7648E+07
u_c_hilo	1.6641E+06	2.6695E+06	1.8731E+06	2.7135E+06
u_c_lohi	1.7626E+03	2.7639E+03	1.8400E+03	2.7380E+03
u_c_lolo	1.7265E+02	2.7196E+02	1.8169E+02	2.6773E+02
u_i_hihi	4.1277E+06	3.6175E+06	3.2489E+06	6.5511E+06
u_i_hilo	4.6574E+05	4.0982E+05	3.2768E+05	7.1039E+05
u_i_lohi	4.2063E+02	3.8518E+02	3.2094E+02	6.9389E+02
u_i_lolo	3.4820E+01	4.0810E+01	3.1040E+01	6.7940E+01
u_s_hihi	9.8003E+06	1.5599E+07	1.0826E+07	1.6694E+07
u_s_hilo	8.2527E+05	1.3726E+06	9.9935E+05	1.6607E+06
u_s_lohi	8.7390E+02	1.3767E+03	1.0135E+03	1.6190E+03
u_s_hilo	8.5660E+01	1.4440E+02	1.0283E+02	1.7043E+02

Table 6.11: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic Without Fault Tolerance (2048 × 64 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	1	0.6167	0.4435	0.7590
u_c_hilo	1	0.6168	0.4290	0.7624
u_c_lohi	1	0.6111	0.4371	0.7585
u_c_lolo	1	0.6196	0.4448	0.7592
u_i_hihi	0.3781	0.5816	0.3892	0.7698
u_i_hilo	0.3522	0.5679	0.4219	0.7797
u_i_lohi	0.3855	0.5897	0.4209	0.7731
u_i_lolo	0.4494	0.5687	0.4347	0.7696
u_s_hihi	0.0953	0.5808	0.3031	0.8297
u_s_hilo	0.1208	0.6029	0.3270	0.8206
u_s_lohi	0.1094	0.6145	0.3562	0.8257
u_s_hilo	0.1083	0.6087	0.3470	0.8312

In this simulation, the 8 number of machines is failed. The machine numbers are 10, 3, 4, 15, 1, 8, 16 and 6. The machines are failing after 165, 176, 182, 188, 234, 314, 338 and 370 task respectively. Table 6.12, Table 6.13, Table 6.14 and 6.15 show the total number of task failed in MET, MCT, min-min and max-min heuristic.

Table 6.12: Total Number of Tasks Failed in MET Heuristic (512 × 16 Instances)

Instances / Machine Number	u_c_hihi	u_c_hilo	u_c_lohi	u_c_lolo	u_i_hihi	u_i_hilo	u_i_lohi	u_i_lolo	u_s_hihi	u_s_hilo	u_s_lohi	u_s_lolo
10	0	0	0	0	23	23	30	23	26	13	17	21
3	0	0	0	0	22	19	22	19	0	0	0	0
4	0	0	0	0	20	15	16	22	27	24	14	12
15	0	0	0	0	24	16	27	24	0	0	0	0
1	279	279	279	279	14	9	15	14	144	153	145	149
8	0	0	0	0	6	14	11	15	18	17	13	11
16	0	0	0	0	9	13	13	4	7	5	15	13
6	0	0	0	0	15	16	8	8	7	11	6	7

Table 6.13: Total Number of Tasks Failed in MCT Heuristic (512 × 16 Instances)

Instances / Machine Number	u_c_hihi	u_c_hilo	u_c_lohi	u_c_lolo	u_i_hihi	u_i_hilo	u_i_lohi	u_i_lolo	u_s_hihi	u_s_hilo	u_s_lohi	u_s_lolo
10	18	17	13	19	26	21	20	21	16	23	24	26
3	26	25	30	25	23	19	24	18	22	18	20	21
4	18	22	23	19	15	22	16	23	28	22	24	18
15	16	21	15	24	19	21	25	17	19	27	18	22
1	57	30	43	29	12	18	22	12	20	20	23	21
8	11	9	8	9	9	12	11	13	18	13	15	14
16	10	12	10	14	9	11	15	10	13	8	18	12
6	6	9	8	6	13	10	10	11	10	11	12	6

Table 6.14: Total Number of Tasks Failed in Min-Min Heuristic (512 × 16 Instances)

Instances / Machine Number	u_c_hihi	u_c_hilo	u_c_lohi	u_c_lolo	u_i_hihi	u_i_hilo	u_i_lohi	u_i_lolo	u_s_hihi	u_s_hilo	u_s_lohi	u_s_lolo
10	10	14	10	14	22	21	21	22	29	24	22	28
3	32	36	34	34	22	21	22	22	29	27	28	24
4	22	28	25	29	18	20	21	20	25	27	23	22
15	6	9	5	9	20	18	22	21	4	7	4	7
1	86	49	85	50	19	17	17	18	53	33	54	32
8	8	10	7	10	11	13	11	13	16	16	16	14
16	4	5	3	5	14	10	11	11	12	12	14	15
6	4	5	3	5	11	9	8	8	7	10	8	9

Table 6.15: Total Number of Tasks Failed in Max-Min Heuristic (512 × 16 Instances)

Instances / Machine Number	u_c_hihi	u_c_hilo	u_c_lohi	u_c_lolo	u_i_hihi	u_i_hilo	u_i_lohi	u_i_lolo	u_s_hihi	u_s_hilo	u_s_lohi	u_s_lolo
10	12	16	13	17	21	24	22	24	25	23	26	25
3	29	35	32	33	19	22	20	22	27	28	30	26
4	23	26	23	28	22	21	17	23	23	19	21	22
15	7	11	7	11	25	21	27	24	7	9	5	9
1	93	44	90	44	14	20	18	18	53	31	53	30
8	8	11	8	10	13	10	12	9	15	13	12	13
16	3	6	3	6	14	13	10	10	10	13	13	13
6	8	8	6	8	11	8	14	9	13	9	10	10

6.5.5 Results of MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance

The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 512 $\times$ 16 data sets are shown in Table 6.16 and Table 6.17 respectively. The graphical representation of makespan and machine utilisation for MET, MCT, min-min and max-min (without fault tolerance and with fault tolerance) is shown in Figure 6.6, Figure 6.10, Figure 6.7, Figure 6.11, Figure 6.8, Figure 6.12, Figure 6.9 and Figure 6.13 respectively. The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 1024 $\times$ 32 data sets are shown in Table 6.18 and Table 6.19 respectively. The graphical representation of makespan and machine utilisation for MET, MCT, min-min and max-min (without fault tolerance and with fault tolerance) is shown in Figure 6.14, Figure 6.18, Figure 6.15, Figure 6.19, Figure 6.16, Figure 6.20, Figure 6.17 and Figure 6.21 respectively. The comparison of makespan and machine utilisation for MET, MCT, min-min and max-min heuristics using 2048 $\times$ 64 data sets are shown in Table 6.20 and Table 6.21 respectively. The graphical representation of makespan and machine utilisation for MET, MCT, min-min and max-min (without fault tolerance and with fault tolerance) is shown in Figure 6.22, Figure 6.26, Figure 6.23, Figure 6.27, Figure 6.24, Figure 6.28, Figure 6.25 and Figure 6.29 respectively.

Table 6.16: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (512 × 16 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	5.3052E+07	2.1794E+07	2.1407E+07	1.9058E+07
u_c_hilo	8.7844E+05	3.1088E+05	3.2471E+05	2.8025E+05
u_c_lohi	1.6256E+06	7.3675E+05	6.7661E+05	6.2904E+05
u_c_lolo	2.8962E+04	1.0175E+04	1.0801E+04	9.1983E+03
u_i_hihi	8.5274E+06	9.2631E+06	8.7481E+06	1.2108E+07
u_i_hilo	1.6403E+05	1.6698E+05	1.5813E+05	2.1763E+05
u_i_lohi	2.7597E+05	3.0498E+05	2.9737E+05	3.9092E+05
u_i_lolo	5.6228E+03	5.2437E+03	5.6108E+03	7.3042E+03
u_s_hihi	3.0105E+07	1.5219E+07	1.6121E+07	1.6855E+07
u_s_hilo	3.1881E+05	2.3047E+05	2.3919E+05	2.4758E+05
u_s_lohi	8.6459E+05	4.4428E+05	4.4101E+05	4.8025E+05
u_s_hilo	1.2084E+04	8.1171E+03	8.1789E+03	9.0163E+03

Table 6.17: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (512 × 16 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	0.7088	0.6424	0.5228	0.7969
u_c_hilo	0.8071	0.6489	0.5610	0.7898
u_c_lohi	0.7056	0.6320	0.5295	0.7793
u_c_lolo	0.8119	0.6568	0.5698	0.7941
u_i_hihi	0.4901	0.6285	0.4971	0.7865
u_i_hilo	0.5080	0.6339	0.5523	0.8001
u_i_lohi	0.5091	0.6256	0.4991	0.8167
u_i_lolo	0.4949	0.6471	0.5296	0.7992
u_s_hihi	0.3032	0.6210	0.4965	0.7831
u_s_hilo	0.4960	0.6264	0.5398	0.7891
u_s_lohi	0.3105	0.6060	0.5112	0.7913
u_s_hilo	0.4637	0.6227	0.5504	0.7809

Table 6.18: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (1024 × 32 Instances)

Instances	**MET**	**MCT**	**Min-Min**	**Max-Min**
u_c_hihi	7.1204E+07	9.0368E+07	9.2542E+07	7.7707E+07
u_c_hilo	7.3878E+06	9.3558E+06	9.2014E+06	7.5687E+06
u_c_lohi	6.8098E+03	9.1600E+03	9.2120E+03	7.4429E+03
u_c_lolo	7.8480E+02	9.2257E+02	9.4887E+02	7.7871E+02
u_i_hihi	1.3968E+07	1.5616E+07	1.3936E+07	2.0882E+07
u_i_hilo	1.4525E+06	1.5199E+06	1.2984E+06	2.0121E+06
u_i_lohi	1.6435E+03	1.5039E+03	1.3579E+03	2.2663E+03
u_i_lolo	1.4735E+02	1.5151E+02	1.4079E+02	2.2828E+02
u_s_hihi	4.0502E+07	5.5970E+07	6.1872E+07	4.6608E+07
u_s_hilo	3.7105E+06	5.4920E+06	5.9335E+06	4.4402E+06
u_s_lohi	3.6559E+03	5.4971E+03	5.7314E+03	4.1854E+03
u_s_hilo	4.0626E+02	5.7442E+02	6.0615E+02	4.3121E+02

Table 6.19: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (1024 × 32 Instances)

Instances	**MET**	**MCT**	**Min-Min**	**Max-Min**
u_c_hihi	0.6000	0.5794	0.5032	0.6889
u_c_hilo	0.5580	0.5668	0.5046	0.6879
u_c_lohi	0.5729	0.5714	0.5013	0.6886
u_c_lolo	0.5558	0.5758	0.5003	0.6861
u_i_hihi	0.4437	0.5609	0.4630	0.7399
u_i_hilo	0.4117	0.5570	0.4639	0.7451
u_i_lohi	0.3956	0.5855	0.4731	0.7368
u_i_lolo	0.4196	0.5785	0.4445	0.7504
u_s_hihi	0.1337	0.5671	0.4695	0.7552
u_s_hilo	0.1447	0.5591	0.4797	0.7195
u_s_lohi	0.1361	0.5439	0.4796	0.7393
u_s_hilo	0.1226	0.5527	0.4776	0.7542

Table 6.20: Makespan Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (2048 × 64 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	3.5770E+07	7.2217E+07	6.7673E+07	6.8052E+07
u_c_hilo	3.5821E+06	7.1534E+06	6.6778E+06	6.5929E+06
u_c_lohi	3.7223E+03	7.3549E+03	6.7535E+03	6.8577E+03
u_c_lolo	3.7123E+02	7.2791E+02	6.6689E+02	6.6836E+02
u_i_hihi	5.2893E+06	5.6261E+06	4.7571E+06	1.1040E+07
u_i_hilo	5.7573E+05	5.7434E+05	4.8487E+05	1.2574E+06
u_i_lohi	5.2275E+02	5.6427E+02	4.8307E+02	9.4297E+02
u_i_lolo	6.1960E+01	5.7580E+01	4.6520E+01	9.6640E+01
u_s_hihi	2.2545E+07	4.0392E+07	3.9095E+07	3.1368E+07
u_s_hilo	1.7650E+06	3.7840E+06	3.7126E+06	3.0262E+06
u_s_lohi	1.7365E+03	3.9021E+03	3.7054E+03	3.0513E+03
u_s_hilo	2.1785E+02	4.0979E+02	3.8993E+02	3.1886E+02

Table 6.21: Machine Utilisation Values for MET, MCT, Min-Min and Max-Min Heuristic With Fault Tolerance (2048 × 64 Instances)

Instances	MET	MCT	Min-Min	Max-Min
u_c_hihi	0.5709	0.5957	0.5050	0.6972
u_c_hilo	0.5523	0.5871	0.5044	0.6988
u_c_lohi	0.5494	0.5826	0.5101	0.6976
u_c_lolo	0.5600	0.5799	0.5014	0.7003
u_i_hihi	0.4046	0.5313	0.4024	0.6126
u_i_hilo	0.3901	0.5662	0.4202	0.6237
u_i_lohi	0.4418	0.5793	0.4261	0.7409
u_i_lolo	0.3627	0.5607	0.4290	0.7594
u_s_hihi	0.0801	0.5746	0.4498	0.7932
u_s_hilo	0.0935	0.5848	0.4459	0.7846
u_s_lohi	0.0918	0.5607	0.4520	0.7908
u_s_hilo	0.0809	0.5610	0.4550	0.7906

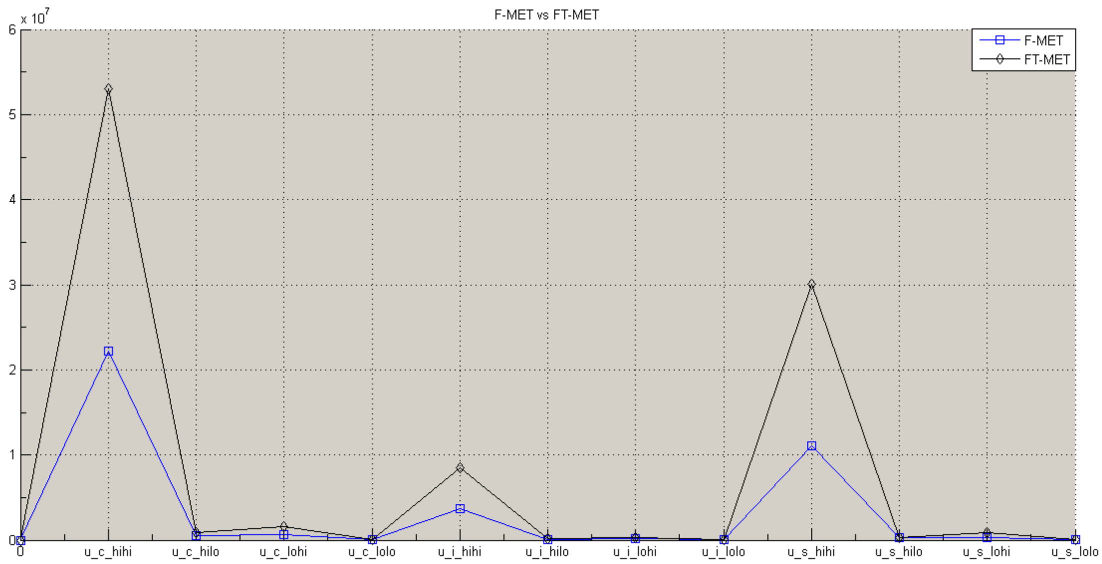

Figure 6.6: Makespan for MET Without Fault Tolerance vs Makespan for MET With Fault Tolerance (512 $\times$ 16 Instances)

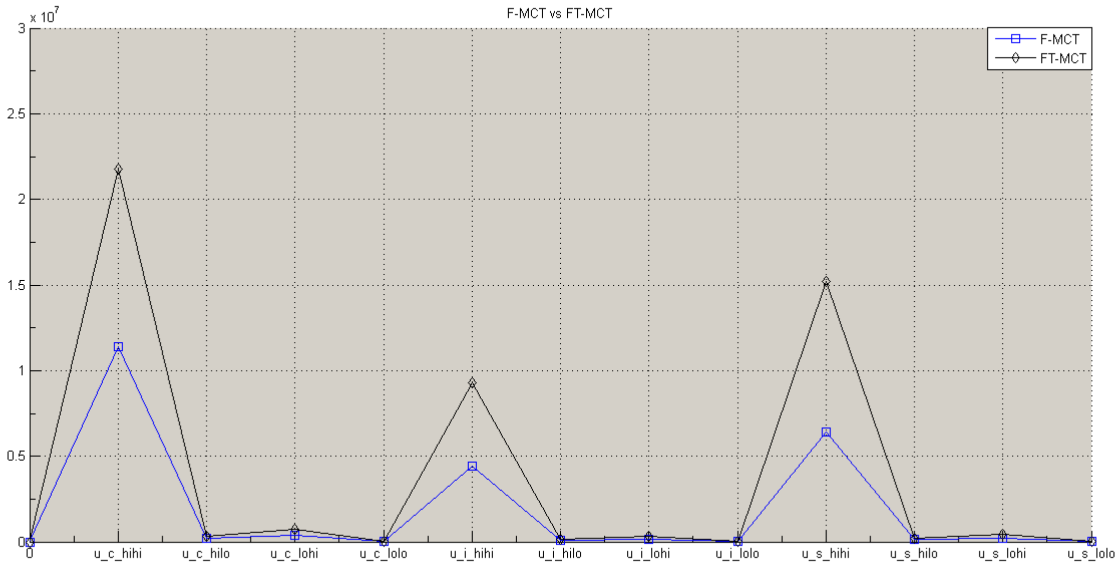

Figure 6.7: Makespan for MCT Without Fault Tolerance vs Makespan for MCT With Fault Tolerance (512 $\times$ 16 Instances)

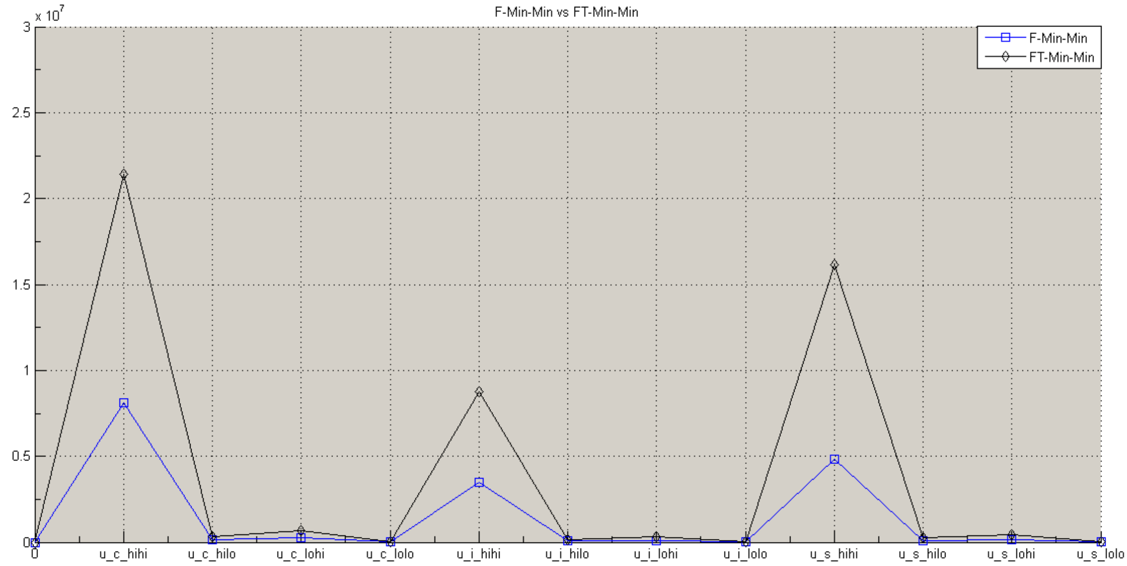

Figure 6.8: Makespan for Min-Min Without Fault Tolerance vs Makespan for Min-Min With Fault Tolerance (512 × 16 Instances)

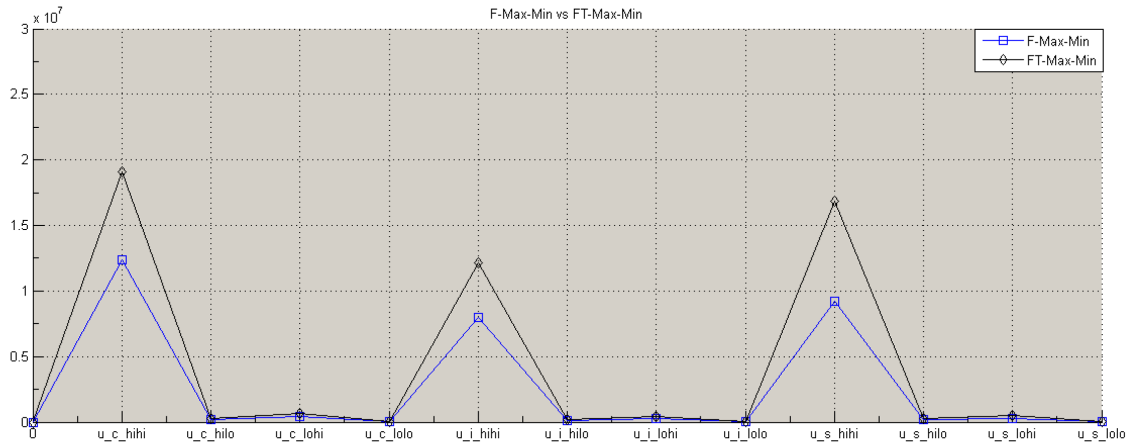

Figure 6.9: Makespan for Max-Min Without Fault Tolerance vs Makespan for Max-Min With Fault Tolerance (512 × 16 Instances)

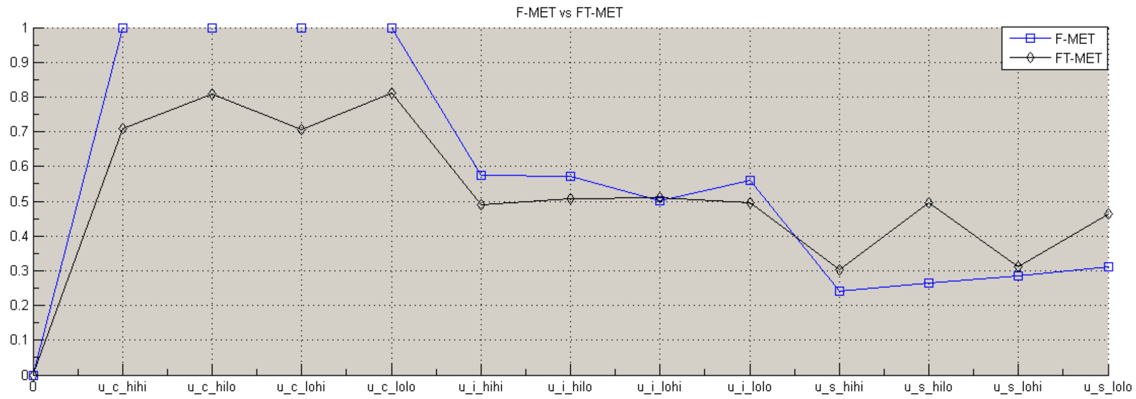

Figure 6.10: Machine Utilisation for MET Without Fault Tolerance vs Machine Utilisation for MET With Fault Tolerance (512 × 16 Instances)

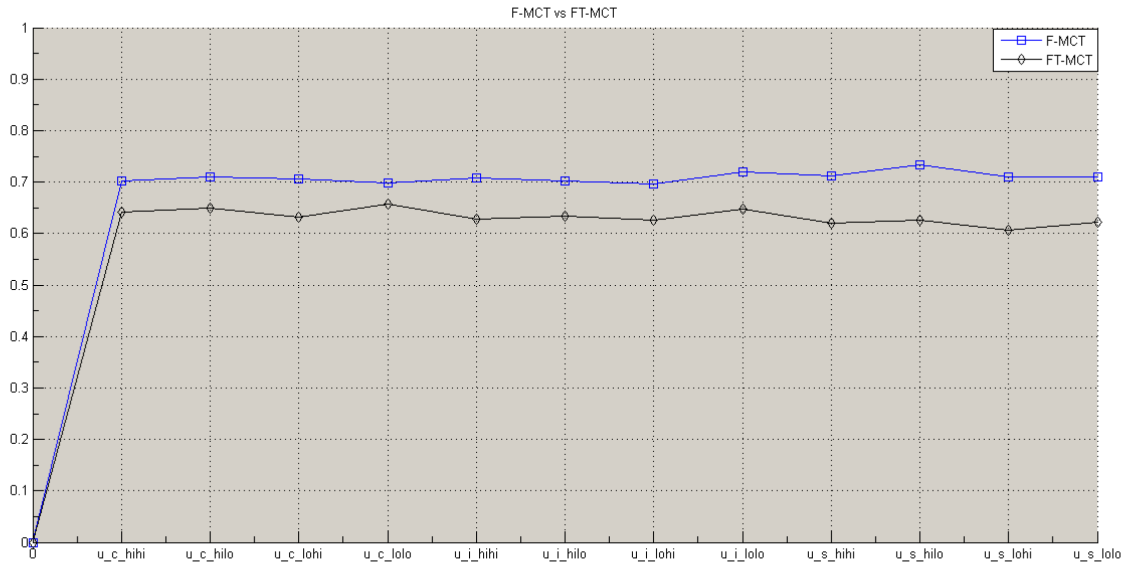

Figure 6.11: Machine Utilisation for MCT Without Fault Tolerance vs Machine Utilisation for MCT With Fault Tolerance (512 × 16 Instances)

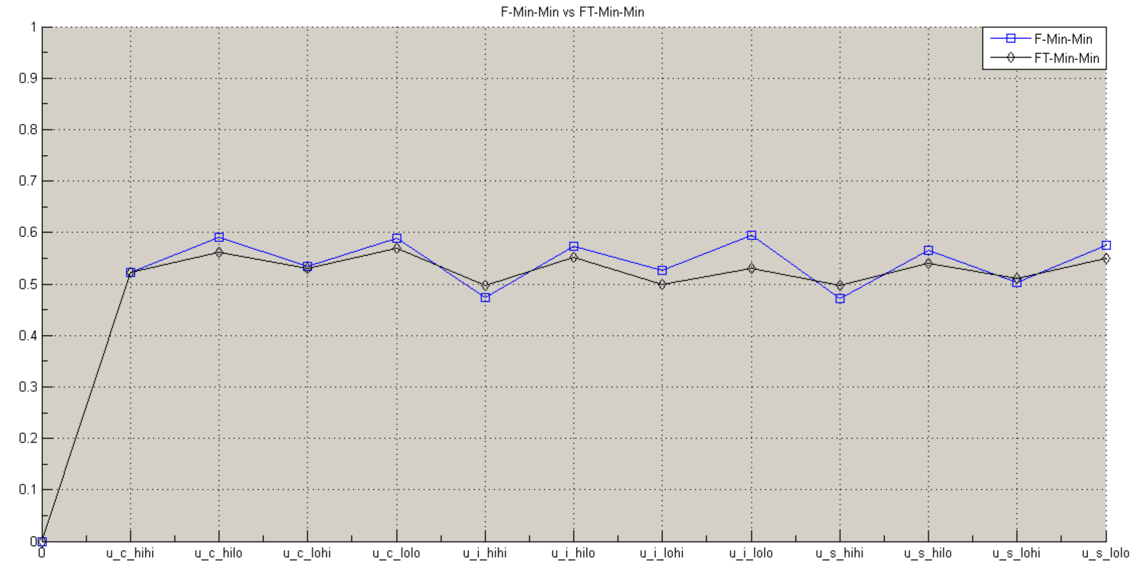

Figure 6.12: Machine Utilisation for Min-Min Without Fault Tolerance vs Machine Utilisation for Min-Min With Fault Tolerance (512 × 16 Instances)

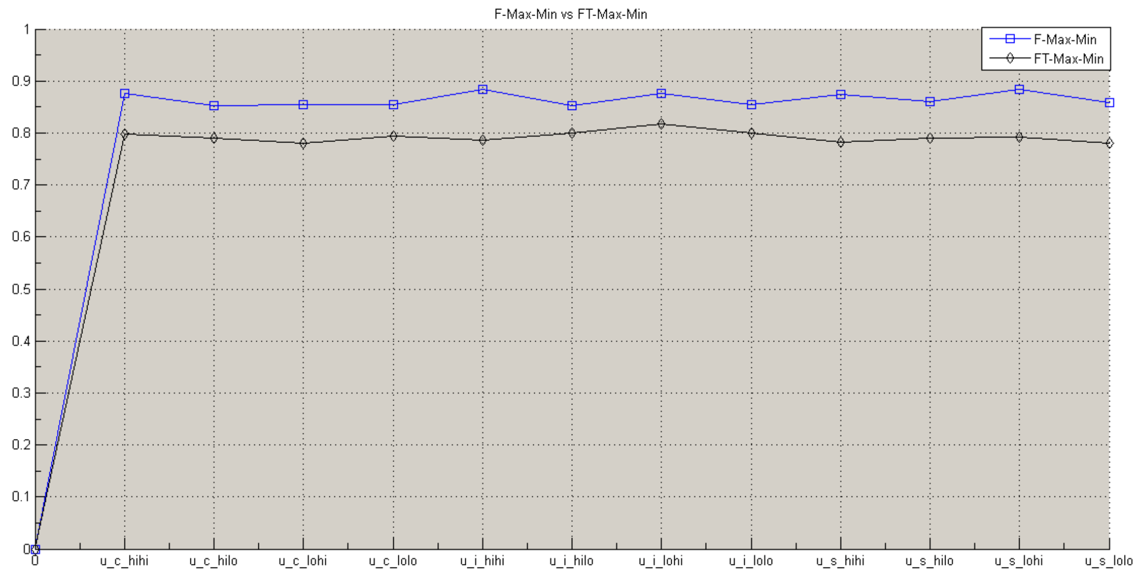

Figure 6.13: Machine Utilisation for Max-Min Without Fault Tolerance vs Machine Utilisation for Max-Min With Fault Tolerance (512 × 16 Instances)

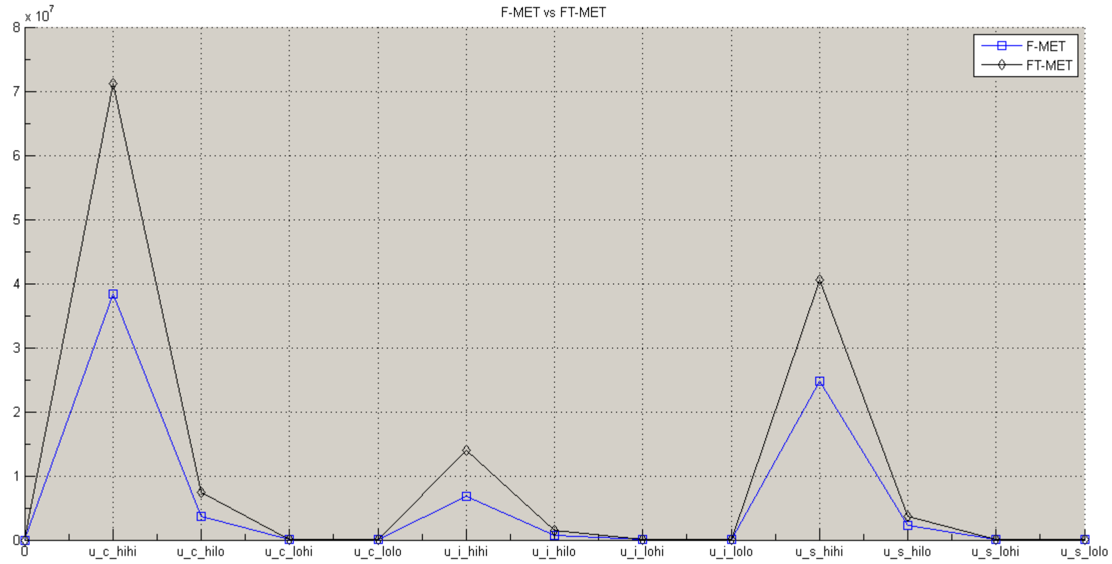

Figure 6.14: Makespan for MET Without Fault Tolerance vs Makespan for MET With Fault Tolerance (1024 × 32 Instances)

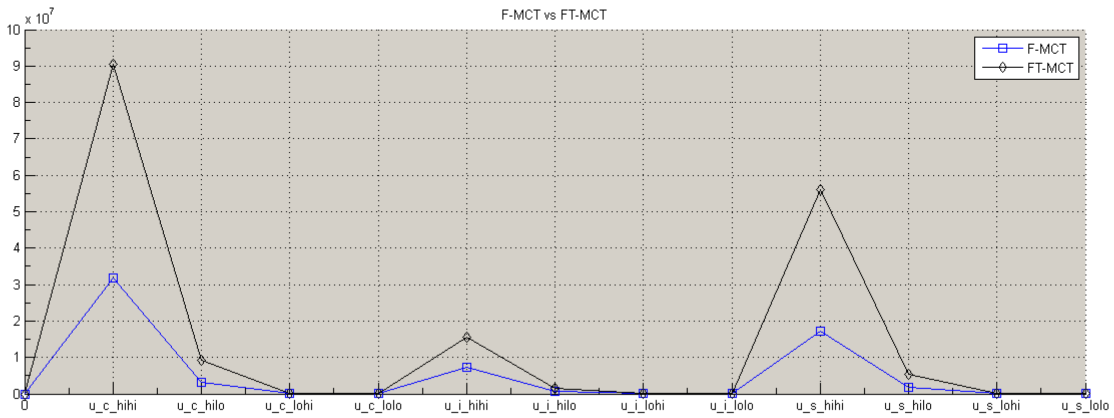

Figure 6.15: Makespan for MCT Without Fault Tolerance vs Makespan for MCT With Fault Tolerance (1024 × 32 Instances)

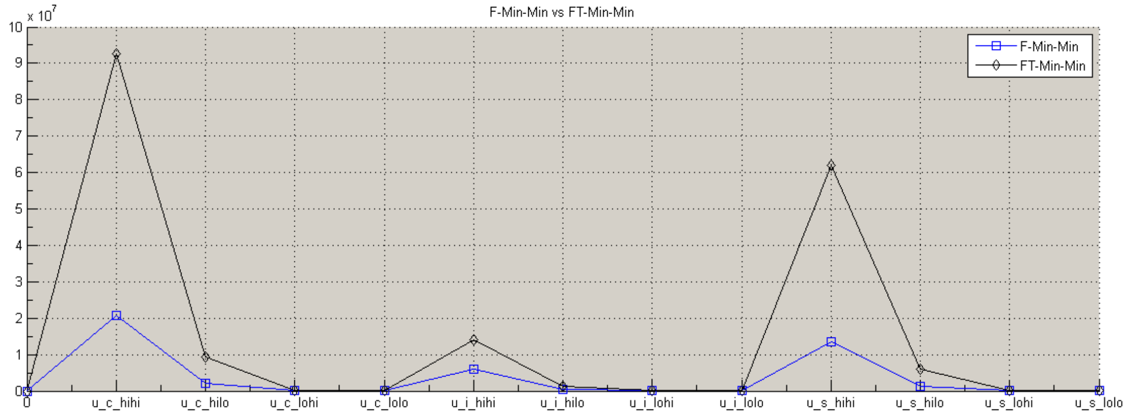

Figure 6.16: Makespan for Min-Min Without Fault Tolerance vs Makespan for Min-Min With Fault Tolerance (1024 × 32 Instances)

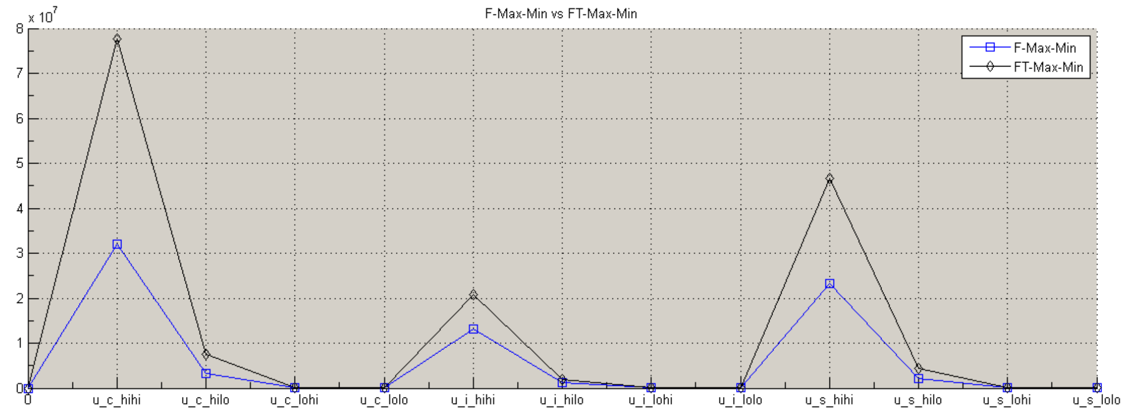

Figure 6.17: Makespan for Max-Min Without Fault Tolerance vs Makespan for Max-Min With Fault Tolerance (1024 × 32 Instances)

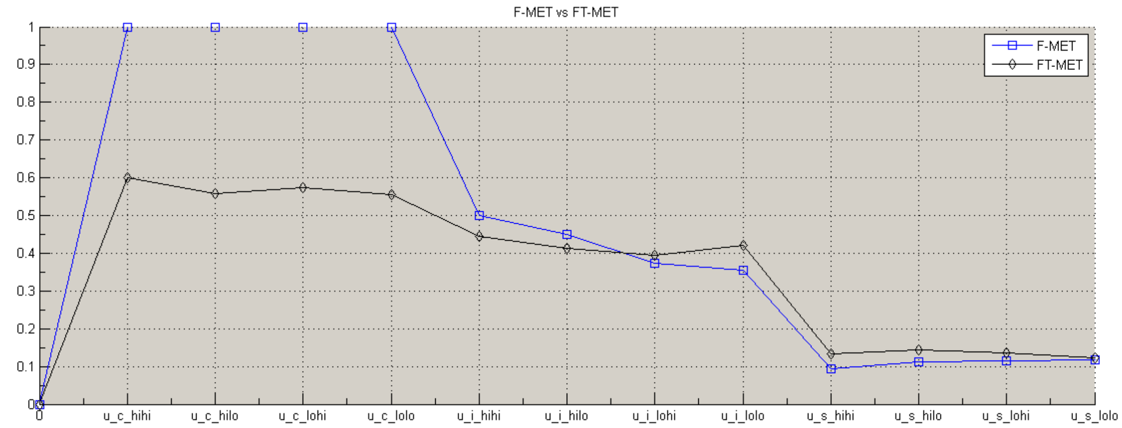

Figure 6.18: Machine Utilisation for MET Without Fault Tolerance vs Machine Utilisation for MET With Fault Tolerance (1024 × 32 Instances)

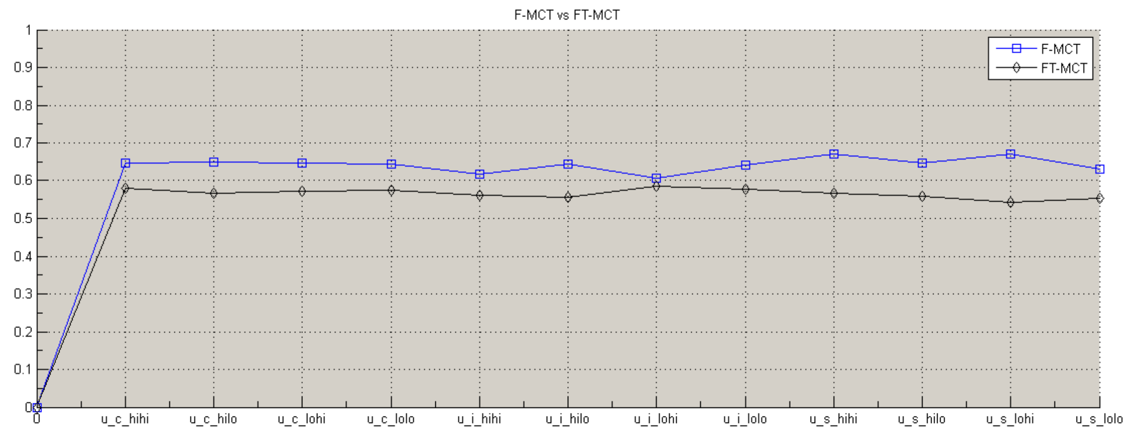

Figure 6.19: Machine Utilisation for MCT Without Fault Tolerance vs Machine Utilisation for MCT With Fault Tolerance (1024 × 32 Instances)

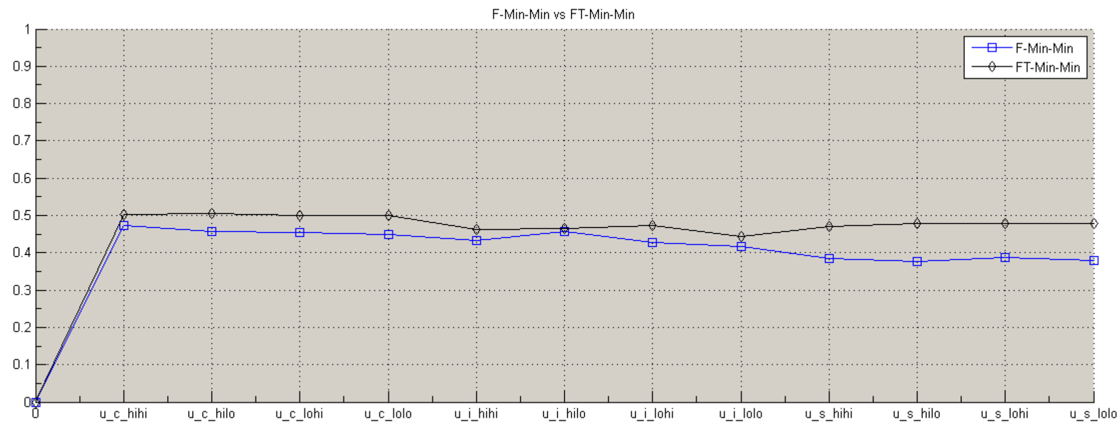

Figure 6.20: Machine Utilisation for Min-Min Without Fault Tolerance vs Machine Utilisation for Min-Min With Fault Tolerance (1024 × 32 Instances)

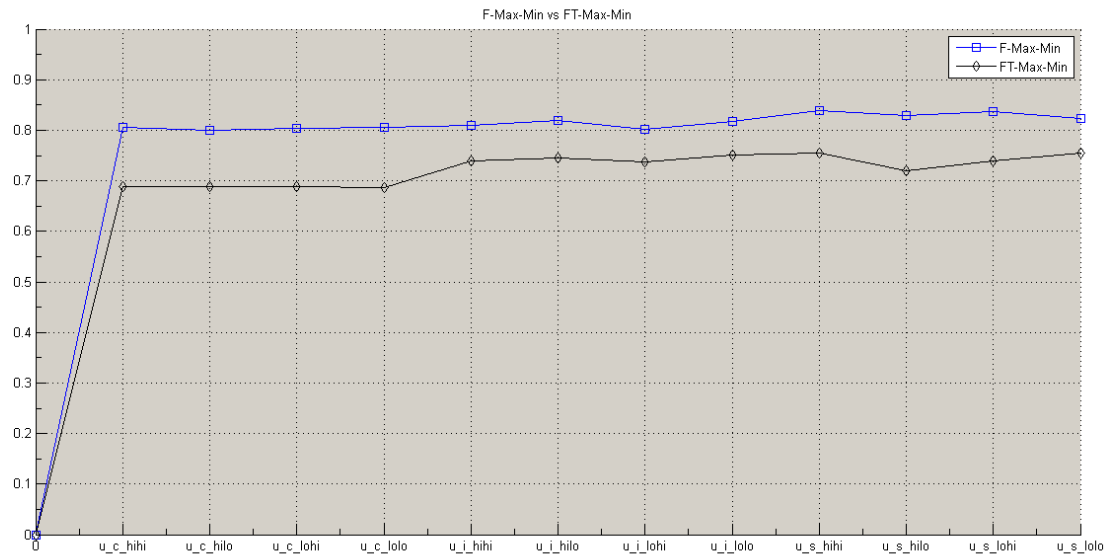

Figure 6.21: Machine Utilisation for Max-Min Without Fault Tolerance vs Machine Utilisation for Max-Min With Fault Tolerance (1024 × 32 Instances)

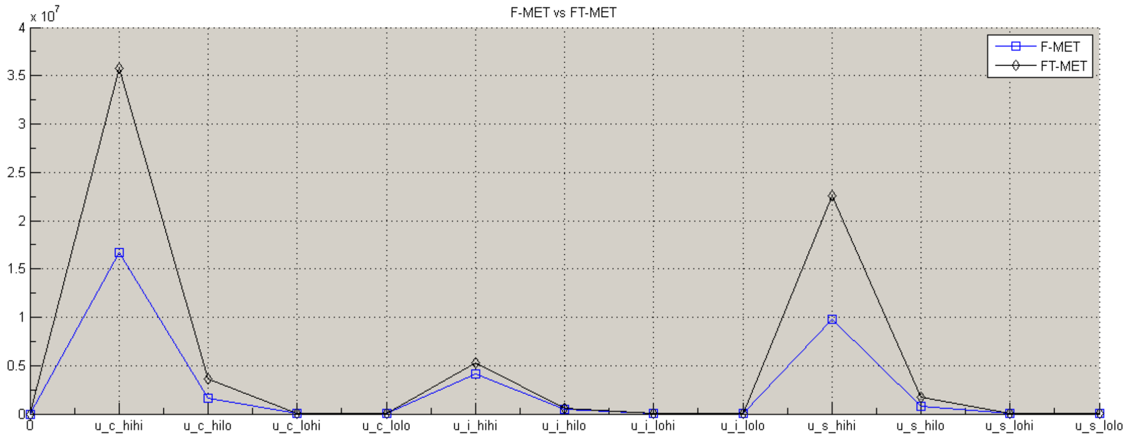

Figure 6.22: Makespan for MET Without Fault Tolerance vs Makespan for MET With Fault Tolerance (2048 $\times$ 64 Instances)

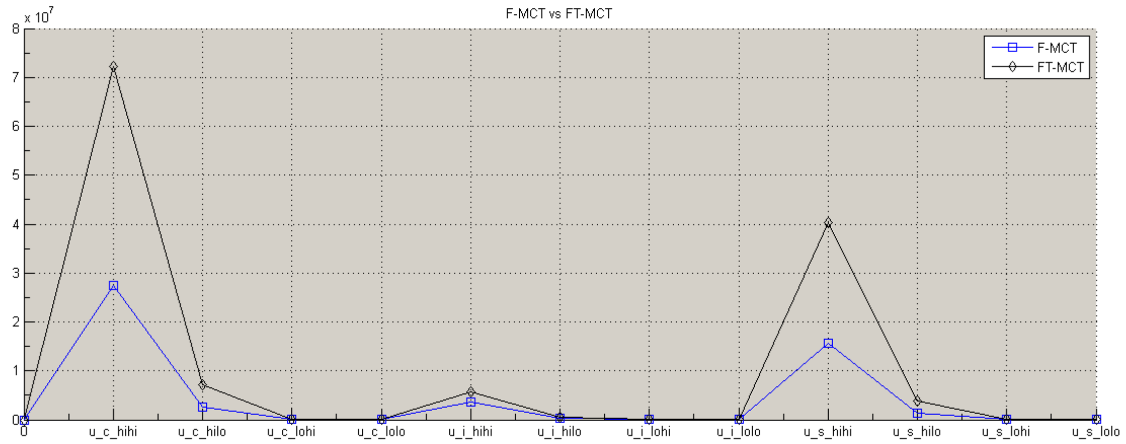

Figure 6.23: Makespan for MCT Without Fault Tolerance vs Makespan for MCT With Fault Tolerance (2048 $\times$ 64 Instances)

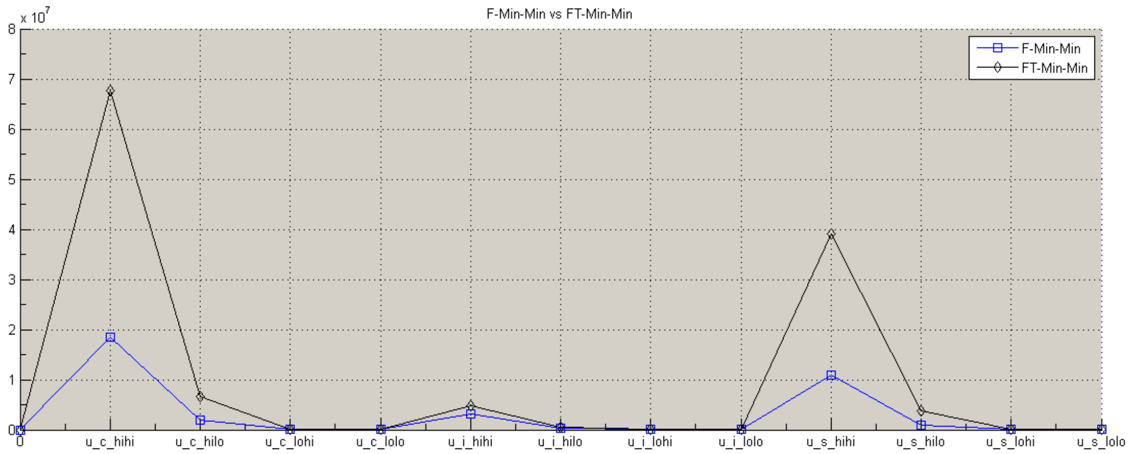

Figure 6.24: Makespan for Min-Min Without Fault Tolerance vs Makespan for Min-Min With Fault Tolerance (2048 × 64 Instances)

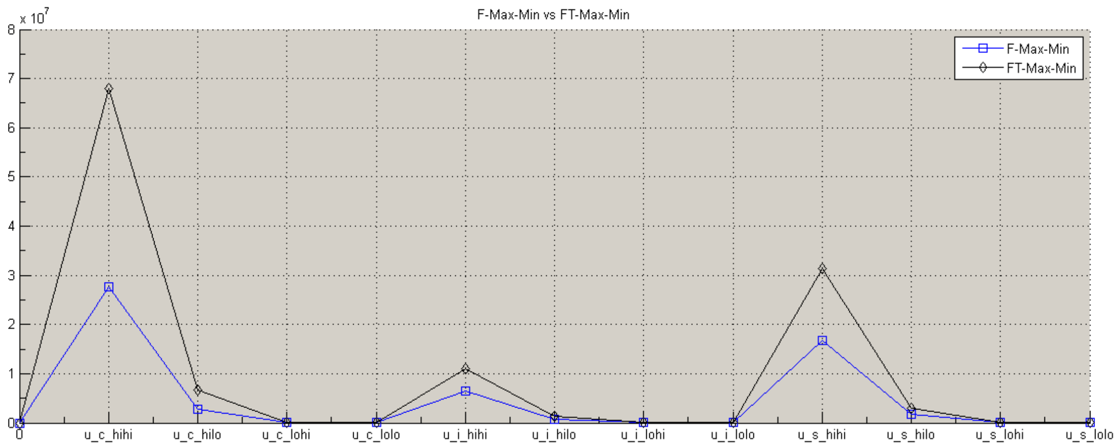

Figure 6.25: Makespan for Max-Min Without Fault Tolerance vs Makespan for Max-Min With Fault Tolerance (2048 × 64 Instances)

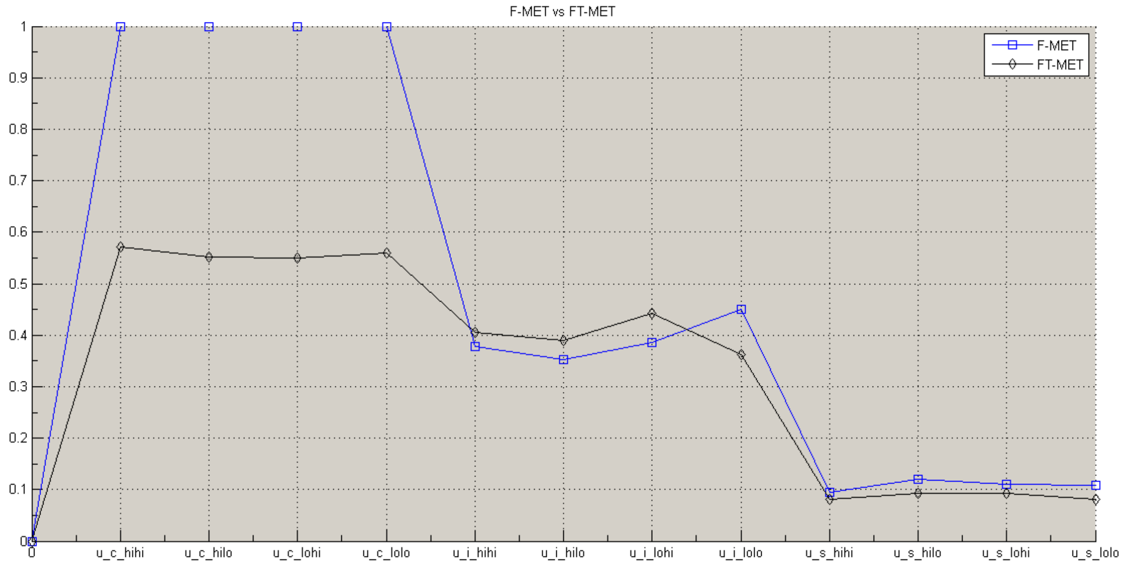

Figure 6.26: Machine Utilisation for MET Without Fault Tolerance vs Machine Utilisation for MET With Fault Tolerance (2048 × 64 Instances)

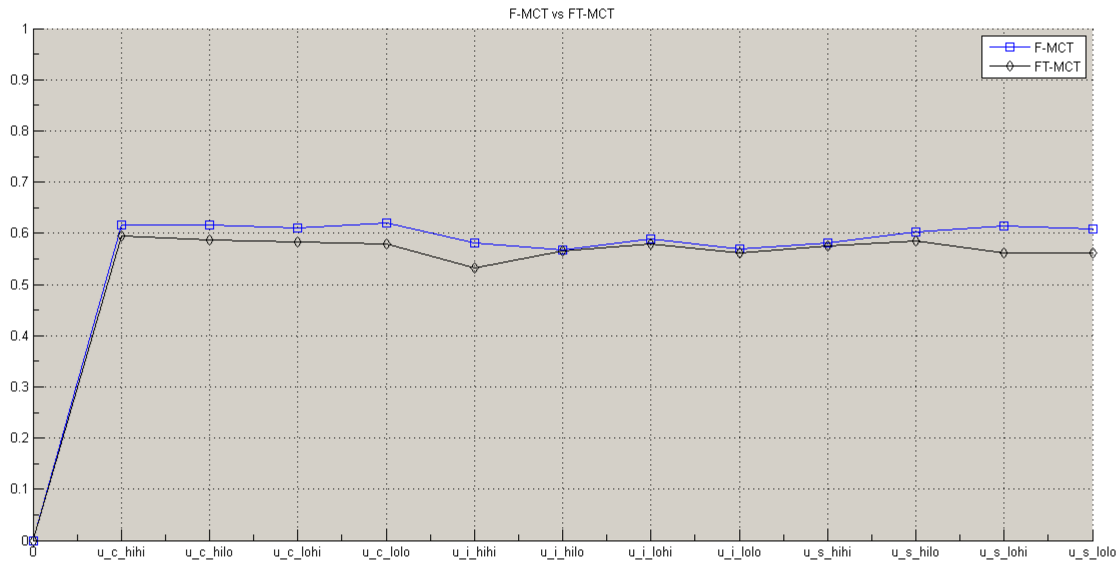

Figure 6.27: Machine Utilisation for MCT Without Fault Tolerance vs Machine Utilisation for MCT With Fault Tolerance (2048 × 64 Instances)

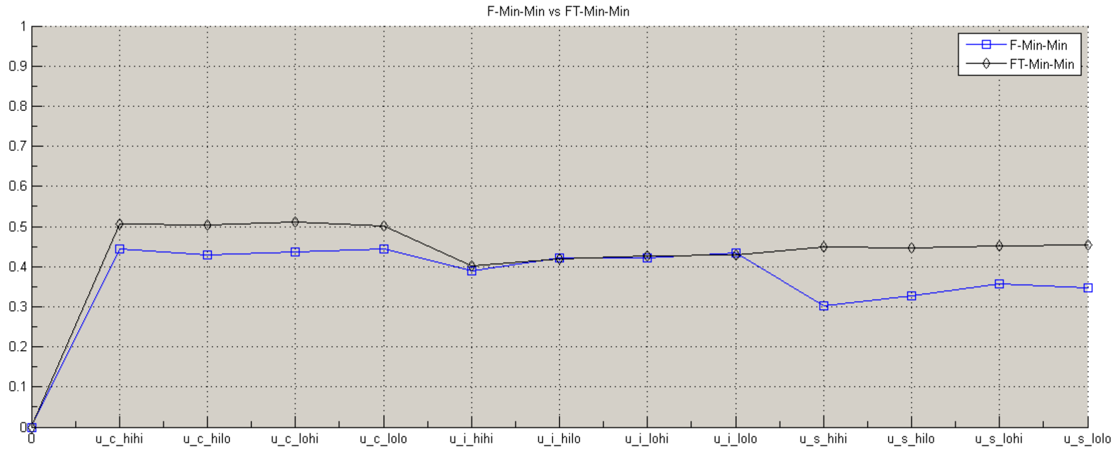

Figure 6.28: Machine Utilisation for Min-Min Without Fault Tolerance vs Machine Utilisation for Min-Min With Fault Tolerance (2048 × 64 Instances)

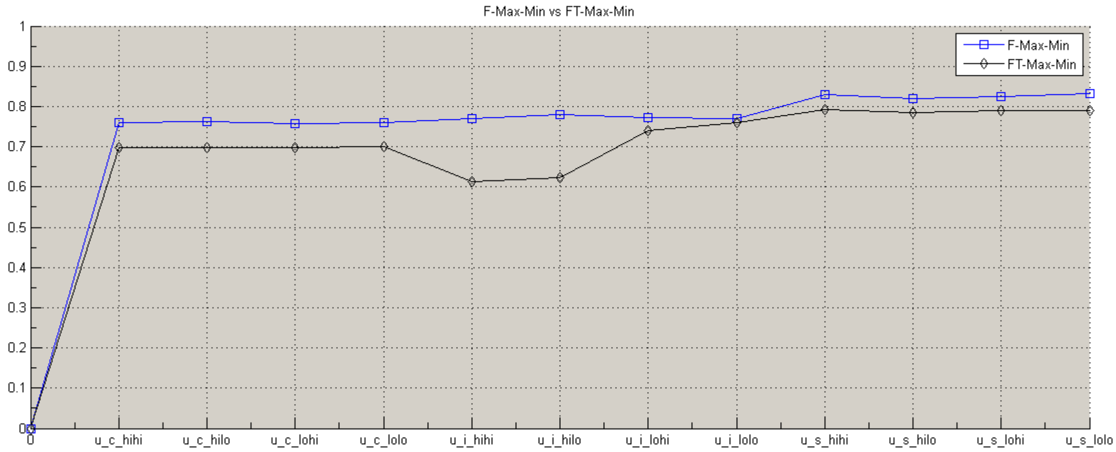

Figure 6.29: Machine Utilisation for Max-Min Without Fault Tolerance vs Machine Utilisation for Max-Min With Fault Tolerance (2048 × 64 Instances)

6.6 Summary

In this chapter, we have implemented the proposed heuristics. The heuristics are compared with the existing heuristics and results are briefly discussed.

Chapter 7

Conclusion and Future Work

Task scheduling and fault tolerance are two important issues in the recent grid computing scenario. Efficient scheduling heuristics are needed to utilize the resource effectively and reduce the overall completion time. The main goal of grid task scheduling is to increase the throughput based on availability of resources. In this thesis, three batch mode heuristics are proposed and compared with min-min and max-min heuristic. Apart from that, four fault tolerant scheduling are proposed based on the existing heuristics. The experimental results show that SIM^2, TSA and RRTS show better performance than the other existing heuristics. The proposed fault tolerant heuristics are experimented and compared with the existing heuristics. It shows better performance than the other existing heuristics.

In the future, we can extend our scheduling approach by using communication cost between tasks, deadline of tasks, dynamic priority and security mechanisms. We can extend our fault tolerant approach to implement some more real life aspects like transient fault, intermittent fault and benign fault.

Bibliography

1. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999.

2. BOINC, *http://boinc.berkeley.edu/*, Accessed on 11^{th} May 2013.

3. M. Murshed, R. Buyya and David Abramson, GridSim: A Toolkit for the Modeling and Simulation of Global Grids, *Technical Report*, Monash University, Australia, 2001.

4. R. Buyya, *High Performance Cluster Computing*, Pearson Education, 2008.

5. I. Foster, C. Kesselman and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of High Performance Computing Applications*, Vol. 15, No. 3, pp. 200 222, 2001.

6. D. Heap, Scorpion: Simplifying the Corporate IT Infrastructure, *IBM Research White Paper*, 2000.

7. A. Chakrabarti, *Grid Computing Security*, Springer-Verlag Berlin Heidelberg, 2007.

8. Types of Grids, *http://thegridweblog.blogspot.in/2005/10/types-of-grids.html*, Accessed on 19^{th} May 2013.

9. F. F. Rivera, M. Bubak and A. G. Tato and R. Doallo, Grid Characteristics and Uses: A Grid Definition, *Lecture Notes in Computer Science, Springer*, Vol. 2970, pp. 291 298, 2004.

10. SETI@home, *http://setiathome.berkeley.edu/*, Accessed on 28^{th} April 2013.

11. Milkyway@home, *http://milkyway.cs.rpi.edu/milkyway/*, Accessed on 28^{th} April 2013.

12. Einstein@home, *http://boinc.berkeley.edu/wiki/Einstein@Home*, Accessed on 28^{th} April 2013.

13. R. Buyya and S. Venugopal, A Gentle Introduction to Grid Computing and Technologies, *Computer Society of India Communications*, Vol. 29, No. 1, pp. 9 19, 2005.

14. M. T. Huda, H. W. Schmidt and I. D. Peake, An Agent Oriented Proactive Fault-tolerant Framework for Grid Computing, *Proceedings of the First International Conference on e-Science and Grid Computing (e-Science)*, pp. 311 318, 2005.

15. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, pp. 107 131, 1999.

16. R. Armstrong, D. Hensgen and T. Kidd, The Relative Performance of Various Mapping Algorithms is Independent of Sizable Variances in Run-time Predictions, *Proceedings of Seventh Heterogeneous Computing Workshop (HCW), IEEE*, pp. 79 87, 1988.

17. T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys and B. Yao, 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. 6, pp. 810 - 837, 2001.

18. H. Xiaoshan, S. Xianhe and G. V. Laszewski, QoS Guided Min-Min Heuristic for Grid Task Scheduling, *Journal of Computer Science and Technology*, Vol. 18, No. 4, pp. 442 - 451, 2003.

19. F. Xhafa, L. Barolli and A. Durresi, Batch Mode Scheduling in Grid Systems, *International Journal of Web and Grid Services*, Vol. 3, No. 1, pp. 19 37, 2007.

20. F. Xhafa, L. Barolli and A. Durresi, Immediate Mode Scheduling in Grid Systems, *International Journal of Web and Grid Services*, Vol. 3, No. 2, pp. 219 236, 2007.

21. A. K. Chaturvedi and R. Sahu, New Heuristic for Scheduling of Independent Tasks in Computational Grid, *International Journal of Grid and Distributed Computing*, Vol. 4, No. 3, pp. 25 36, 2011.

22. L. M. Khanli, M. E. Far and A. M. Rahmani, RFOH: A New Fault Tolerant Job Scheduler in Grid Computing, *Second International Conference on Computer Engineering and Applications (ICCEA)*, pp. 422 425, 2010.

23. P. Latchoumy and P. S. A. Khader, Improved Fault Tolerant Job Scheduler for Optimal Resource Utilization in Computational Grid, *International Journal of Computer Applications*, Vol. 48, No. 22, pp. 6 12, 2012.

24. K. Etminani and M. Naghibzadeh, A Min-Min Max-Min Selective Algorithm for Grid Task Scheduling, *3^{rd} IEEE / IFIP International Conference on Internet (ICI), IEEE*, pp. 1 7, 2007.

25. Z. Jinquan, N. Lina and J. Changjun, A Heuristic Scheduling Strategy for Independent Tasks on Grid, *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA), IEEE*, pp. 593 598, 2005.

26. S. Parsa and R. E. Maleki, RASA: A New Grid Task Scheduling Algorithm, *International Journal of Digital Content Technology and its Applications*, Vol. 3, No. 4, pp. 91 99, 2009.

27. T. Kokilavani and D. I. G. Amalarethinam, Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing, *International Journal of Computer Applications*, Vol. 20, No. 2, pp. 43 49, 2011.

28. F. Dong, J. Luo, L. Gao and L. Ge, A Grid Task Scheduling Algorithm Based on QoS Priority Grouping, *Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC), IEEE*, pp. 58 61, 2006.

29. E. U. Munir, J. Li and S. Shi, QoS Sufferage Heuristic for Independent Task Scheduling in Grid, *Information Technology Journal*, Vol. 6, No. 8, pp. 1166 1170, 2007.

30. B. Nazir and T. Khan, Fault Tolerant Job Scheduling in Computational Grid, *2^{nd} International Conference on Emerging Technologies (ICET), IEEE*, pp. 708 713, 2006.

31. N. Upadhyay and M. Misra, Incorporating fault tolerance in ga-based scheduling in grid environment, *World Congress Information and Communication Technologies (WICT)*, pp. 772 - 777, 2011.

32. S. B. Priya, M. Prakash and K. K. Dhawan, Fault tolerance-genetic algorithm for grid task scheduling using check point, *Sixth International Conference on Grid and Cooperative Computing (GCC)*, pp. 676 680, 2007.

33. S. Guo, H. Z. Huang, Z. Wang and Min Xie, Grid service reliability modeling and optimal task scheduling considering fault recovery, *IEEE Transactions on Reliability*, Vol. 60, No. 1, pp. 263 - 274, 2011.

34. D. Nanthiya and P. Keerthika, Load balancing gridsim architecture with fault tolerance, *International Conference on Information Communication and Embedded Systems (ICICES)*, pp. 425 428, 2013.

35. H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, *Proceedings of 9th Heterogeneous Computing Workshop (HCW), IEEE*, pp. 349 363, 2000.

36. A. Abraham, R. Buyya and B. Nath, Natures Heuristics for Scheduling Jobs on Computational Grids, *Proceedings of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM)*, 2000.

37. S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen and S. Ali, Task Execution Time Modeling for Heterogeneous Computing Systems, *Proceedings of Heterogeneous Computing Workshop (HCW), IEEE*, pp. 185 199, 2000.

38. M. Murshed and R. Buyya, Using the GridSim Toolkit for Enabling Grid Computing Education, *Proceedings of the International Conference on Communication Networks and Distributed Systems Modeling and Simulation*, 2002.

39. F. Azzedin and M. Maheswaran, Towards Trust-Aware Resource Management in Grid Computing Systems, *Proceedings of the 2^{nd} IEEE / ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 452 457, 2002.

40. F. Azzedin and M. Maheswaran, Evolving and Managing Trust in Grid Computing Systems, *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1424 1429, 2002.

41. S. Abuelenin, "Trust Based Grid Batch Mode Scheduling Algorithms", *The 8^{th} International Conference on INFOrmatics and Systems (INFO)*, pp. 46 - 54, 2012.

42. B. M. Boghosian and P. V. Coveney, "Scientific Applications of Grid Computing", *Computer Society, IEEE*, 2005.

43. D. Zhu and J. Fan, An Introduction to Aggregation Grid, *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKG), IEEE*, pp. 86 87, 2006.

44. D. Zhu and J. Fan, Aggregation Grid, *Proceedings of the IEEE International Conference on Information Technology (ICIT)*, pp. 357 364, 2007.

45. Z. Gao, S. Luo and D. Ding, A Scheduling Mechanism Considering Simultaneous Running of Grid Tasks and Local Tasks in the Computational Grid, *International Conference on Multimedia and Ubiquitous Engineering (MUE), IEEE*, pp. 1100 1105, 2007.

46. F. Pop, C. Dobre and V. Cristea, Evaluation of Multi-Objective Decentralized Scheduling for Applications in Grid Environment, *4^{th} International Conference on Intelligent Computer Communication and Processing (ICCP), IEEE*, pp. 231 238, 2008.

47. A. Rasooli, M. M. Aghatabar and S. Khorsandi, Introduction of Novel Rule Based Algorithms for Scheduling in Grid Computing Systems, *Second Asia International Conference on Modelling & Simulation (AICMS), IEEE*, pp. 138 143, 2008.

48. Y. Demchenko, C. D. Laat, O. Koeroo and D. Groep, Re-thinking Grid Security Architecture, *Proceedings of Fourth IEEE International Conference on eScience*, pp. 79 86, 2008.

49. X. M. Wen, W. Zhao and F. X. Meng, Research of Grid Scheduling Algorithm Based on P2P_Grid Model, *International Conference on Electronic Commerce and Business Intelligence (ECBI), IEEE*, pp. 41 44, 2009.

50. J. Bagherzadeh and M. M. Adeh, An Improved Ant Algorithm for Grid Scheduling Problem, *Proceedings of the 14^{th} International CSI Computer Conference (CSICC), IEEE*, pp. 323 328, 2009.

51. I. Rodero, F. Guim and J. Corbalan, Evaluation of Coordinated Grid Scheduling Strategies, *11^{th} IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 1 - 10, 2009.

52. F. Xhafa and A. Abraham, Computational Models and Heuristics Methods for Grid Scheduling Problems, *Future Generation Computer Systems, Elsevier*, Vol. 26, pp. 608 621, 2009.

53. B. SenthilKumar, P. Chitra and G. Prakash, Robust Task Scheduling on Heterogeneous Computing Systems using Segmented MaxR-MinCT, *International Journal of Recent Trends in Engineering*, Vol. 1, No. 2, pp.63 65, 2009.

54. H. Decai, Y. Yuan, Z. L. Jun and Z. K. Qin, Research on Tasks Scheduling Algorithms for Dynamic and Uncertain Computing Grid Based on a + bi Connection Number of SPA, *Journal of Software*, Vol. 4, No. 10, pp. 1102 1109, 2009.

55. M. Karimi, A. Bouyer, E. Mohebi and H. Rajabalipour, A Learning-based Approach for Fault Tolerance on Grid Resources Scheduling, *5^{th} IEEE GCC Conference & Exhibition*, pp. 1 5, 2009.

56. D. D. H. Miriam and K. S. Easwarakumar, A Double Min Min Algorithm for Task Metascheduler on Hypercubic P2P Grid Systems, *International Journal of Computer Science Issues*, Vol. 7, No. 5, pp. 8 18, 2010.

57. N. Mansouri. G. Dastghaibyfard and A. Horri, A Novel Job Scheduling Algorithm for Improving Data Grids Performance, *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), IEEE*, pp. 142 147, 2011.

58. M. Bhatia and M. S. Bhatia, Resource Requirement Prioritized Grid Scheduling Model, *International Conference on Emerging Trends in Networks and Computer Communications (ETNCC), IEEE*, pp. 388 -391, 2011.

59. T. Amudha, T. T. Dhivyaprabha, QoS Priority Based Scheduling Algorithm and Proposed Framework for Task Scheduling in a Grid Environment, *IEEE International Conference on Recent Trends in Information Technology (ICRTIT)*, pp. 650 655, 2011.

60. M. Hemamalini, Review on Grid Task Scheduling on Distributed Heterogeneous Environment, *International Journal of Computer Applications*, Vol. 40, No. 2, pp. 24 30, 2012.

61. D. I. G. Amalarethinam and S. V. Kfatheen, Max-Min Average Algorithm for Scheduling Tasks in Grid Computing Systems, *International Journal of Computer Science and Information Technologies*, Vol. 3, No. 2, pp. 3659 3663, 2012.

62. O. M. Elzeki, M. Z. Reshad and M. A. Elsoud, Improved Max-Min Algorithm in Cloud Computing, *International Journal of Computer Applications*, Vol. 50, No. 12, pp. 22 27, 2012.

63. V. Dehalwar, R. K. Baghel and M. Kolhe, Multi-Agent based Public Key Infrastructure for Smart Grid, *The 7^{th} International Conference on Computer Science & Education (ICCSE), IEEE*, pp. 415 418, 2012.

64. S. Thenmozhi, A. Tamilarasi and P. T. Vanathi, A Fault Tolerant Resource Allocation Architecture for Mobile Grid, *Journal of Computer Science*, Vol. 8, No. 6, pp. 978 982, 2012.

65. E. P. Duarte, A. Weber and K. V. O. Fonseca, Distributed Diagnosis of Dynamic Events in Partitionable Arbitrary Topology, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, No. 8, pp. 1415 1426, 2012.

Dissemination

1. Sanjaya Kumar Panda, Sourav Kumar Bhoi and Pabitra Mohan Khilar, "A Semi-Interquartile Min-Min Max-Min (SIM^2) Approach for Grid Task Scheduling", *Proceedings of International Conference on Advances in Computing (ICAdC), Advances in Intelligent Systems and Computing, Springer*, Volume 174, pp. 415 - 421, 2012.

2. Sanjaya Kumar Panda and Pabitra Mohan Khilar, "A Three-Stage Approach for Grid Task Scheduling", 2^{nd} *IEEE International Conference on Parallel Distributed and Grid Computing (PDGC)*, pp. 441 - 446, 2012.

3. Sanjaya Kumar Panda, Sourav Kumar Bhoi and Pabitra Mohan Khilar, "RRTS: A Task Scheduling Algorithm to Minimize Makespan in Grid Environment", *Proceedings of Second International Conference on Internet Computing and Information Communications (ICICIC), Advances in Intelligent and Computing, Springer*, Volume 216, 2013.