A Novel Heuristic for a Class of Independent Tasks in Computational Grids

Pratik Agrawal

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

A Novel Heuristic for a Class of 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

Pratik Agrawal

(Roll No.: 211CS3290)

under the supervision of

Prof. Durga Prasad Mohapatra

And

Prof. Pabitra Mohan Khilar

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.

Certificate

This is to certify that the work in the thesis entitled ***A Novel Heuristic for a Class of Independent Tasks in Computational Grids*** by ***Pratik Agrawal***, bearing ***Roll Number 211CS3290***, is a record of an original research work carried out by him under our 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 ***Software Engineering***. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. Pabitra Mohan Khilar **Prof. Durga Prasad Mohapatra**

Acknowledgment

Foremost, I would like to express my sincere gratitude to my advisors Prof. Durga Prasad Mohapatra and Prof. Pabitra Mohan Khilar for the continuous support of my M.Tech study and research, for their patience, motivation and enthusiasm. Their guidance helped me in the time of research and writing of this thesis. I could not have imagined having better advisors for my M.Tech study.

Besides my advisors, I extend my thanks to our HOD, Prof. A. K. Turuk for his valuable advices and encouragement.

I do acknowledge the academic resources that I have received from NIT Rourkela. I also thank the administrative and technical staff members of the Computer Science Department for their intime support.

My sincere gratitude also goes to Ph.D. Scholar Subhrakanta Panda for constantly guiding me throughout the work.

My sincere thanks also goes to Avijit, Pratik, Meghansh, Vinay, Mukesh, Sanjaya, Dinesh and Godboley for helping me in my work. I would also like to thank Shraddha, Sugandha, Suchitra, Swagtika, Anita, Alina and Anima didi for motivating me time to time.

Last but not the least, I would like to thank my family: my parents Mr. Vijay Agrawal and Mrs. Saroj devi, my sisters Mrs. Aditi Jindal and Mrs. Amita Jain, My brother-in-laws Mr. Satya Prakash Jindal and Mr. Sudhir Jain, and finally my brother Mr. Ankur Agrawal and sister-in-law Mrs. Sonal Agrawal for constantly supporting me throughout my life.

Pratik Agarwal
Roll:211CS2274
Department of Computer Science

Abstract

Scheduling is an essential layer in grid environment. Now-a-days, the computational grids are the important platform for job scheduling. The performance of the computational grids can be improved using an efficient scheduling heuristic. A user submits a job to grid resource broker. The broker is responsible for dividing a job into a number of tasks. It maps the task and the resource to find a perfect match. The main goal is to minimize the processing time and maximize the resource utilization. Mode of scheduling plays the key role in Grid Scheduling. In Grid, mode of scheduling is of two types: *immediate* and *batch mode*. Immediate mode takes one after another task in a serial sequence. But, batch mode takes in a random sequence. Task assignment is mainly based on the mode selection. Task may be assigned to the resource in a batch or as soon as it arrives. In this thesis, we have introduced three immediate mode heuristics such as First-DualMake, Best- DualMake and Worst-DualMake (defined as X-DualMake) and a new mode of heuristic called as intermediate mode (or Multi- batch mode). In our immediate mode scheduling heuristics, jobs are scheduled based on resource idle time. Intermediate mode considers random arrival of task in a multi-batch sequence. Alternatively, arrival of tasks are unknown in this mode. Here, we have taken a range of task arrival for simplicity. This mode is introduced to be a part of the real life aspects. The eight immediate mode heuristics are simulated and the experimental results are discussed. The two existing approaches: Min-Min and Max-Min are experimented with intermediate mode scheduling. We have taken two performance measures: makespan and resource utilization to evaluate the performance.

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

Acronyms

GRB	Grid Resource Broker
GRS	Grid Referral Service
SSI	Single System Image
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
RASA	Resource Aware Scheduling Algorithm
LBMM	Load Balanced Min-Min
QoS	Quality of Service
GIS	Grid Information Service
DQ	Difference Queue
TEQ	TEmporary Queue
CPU	Central Processing Unit
RAM	Random Access Memory
ROM	Read Only Memory
SV	Sufferage Value
RU	Machine (or resource) Utilisation
ARU	Average Machine (or resource) Utilisation

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 M_j using scheduling strategy S
$R(S)$	Machine (or resource) utilisation of scheduling strategy S
$F(S)$	Flow time of scheduling strategy S
$E(S)$	Total execution time of scheduling strategy S
$E(S_{T_i})$	Execution time of T_i using scheduling strategy S
$T_i \rightarrow M_j$	T_i is scheduled to M_j
$T_i \not\rightarrow M_j$	T_i is not scheduled to M_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

Grid computing is a potential technology mainly used for distributed environment. The major issues related with Grid are resource discovery, heterogeneity, fault tolerance and task scheduling. Grid task scheduling is an integrated component of computing which effectively utilizes the idle time of resources [1]. Efficient scheduling algorithm is needed to utilize the resources effectively and reduce the overall completion time. It analyzes the performance of scheduling algorithms from different point of view such as Makespan, execution time, completion time and load balancing. It examines the performance of four scheduling algorithms such as Min-Min, Max-Min, Minimum Completion Time (MCT) and Minimum Execution Time (MET). The conventional Max-Min grid task scheduling algorithm effectively utilizes the resources and minimizes the Makespan than other scheduling algorithms [2] [3]. Scheduling is a NP-Complete problem [2] [4] [1] [5]. One of the goals of Grid task scheduling is to achieve high system throughput (resource utilization) while matching application needs with the available computing resources and balance the load (load balancing).

Grid computing is an innovative extension to parallel and distributed computing technology. It enables sharing, selection and aggregation of geographically distributed resources. This technology is achieving computing resource sharing among participants in a collection of virtual organizations. It provides a virtualized view of the underlying grid resources. Such virtualization also encompasses the security requirements. Therefore, there is a need for virtualization of security semantics to use standardized ways of segmenting security components like authentication, access control, confidentiality etc [5]. In grid technology, security tools are concerned with establishing the identity of users or services (authentication), protecting communications, and determining who is allowed to perform what actions (authorization), as well as with supporting functions such as managing user credentials and maintaining group membership information. The primary motivations behind privacy for grid computing are the need for secure communication (authenticated and confidential) between elements and also the need to support security across organizational boundaries.

Resource Management System (RMS) or Grid Resource Broker (GRB) acts as the central nervous system for a distributed computing grid. It is responsible for resource discovery and reservation, executing consumers tasks, and enforcing the owners security policies for the resource. Heterogeneous Computing refers the interconnection between different high performance machines and high-speed links in a distributed environment. The physical location of Owner, System, Consumer, Application layers are determining the level of security mechanisms that should be put in place. If all layers are physically located behind a secure network, the threat of certain attacks may be mitigated. As such, the level of security and safety mechanisms in place should increase so that imposter parties can be identified and disallowed from secure network in grid. The most prevalent mechanisms of authentication in a Grid Security Infrastructure (GSI) based grid is the certificate based authentication mechanism [6].

We must consider a dynamic environment in which jobs arrive over the time and remove from the task queue at their completion time [4]. Grid task scheduling is not limited to resource utilization but can be extended to the security, central control in administrative domains, real time scheduling and quality of service [3] [5] [7] [8]. Real time has a time limit on computation [11]. Grid resource broker is responsible for allocation of a task to a particular resource which takes less time. It is also responsible for splitting the job into various tasks. Grid applications are divided into number of interdependent subtasks in real applications. Subtask can be processed concurrently to reduce the task execution time [9]. Grid environment consists of many clusters. So, the processors are not only heterogeneous but also the communication is larger. We cannot guarantee the optimum solution but always find solution which is close to optimum [10].

Round Robin algorithm is considered as optimal in time shared environment. RR is a pre-emptive scheduling algorithm. It means the processor released the tasks in the middle of the execution. The tasks which do not have interdependency among them are called Meta tasks. As there is no interdependency, we can execute two tasks in parallel. In other words, if there are two processors in the grid environment then two tasks can be scheduled simultaneously.

Scheduling in grid is not limited to resource utilization but can be stretch out to the quality of service, the security, central control in administrative domains and real time scheduling [8]. Single system Image (SSI) is an illusion to the user. It is designed in such a way that appears as a single resource. When the user submits a job, it is the responsibility of grid resource broker to divide the job into various tasks and assigns to several resources. Further, task can be divided into subtasks and it can be scheduled in parallel. Our end objective is to increase the overall throughput and resource utilization [5]. Also, it is required to break resource idle time and balance the load [10] [7].

List scheduling is of two types: static scheduling and dynamic scheduling. Example of static scheduling is Highest Level First with Estimated Times (HLFET) [5]. It is based on static b-level. It does not guarantee optimal solution [5]. There are two approaches used in scheduling: insertion and non-insertion approach. Because of task interdependency, there is some hole between ordered tasks. If a task is assigned to the first hole, then the approach is called as Insertion approach. If the task is assigned without looking the scheduling hole, then the approach is called as Non-insertion approach [11]. Finally, insertion approach is preferable then non-insertion approach. Insertion Scheduling Heuristic (ISH) and Earliest Time First (ETF) [12] are using non-insertion approach. But, Modified Critical Path (MCP) [8] uses insertion approach

ISH is based on static b-level. It does not guarantee optimal solution because it considers only static b-level. ETF is also based on static b-level. But, it calculates the earliest start-time of a task on each processing elements. Like ISH, it does not guarantee optimal solution MCP is assigning the priority based on As Late As Possible (ALAP). Dynamic Level Scheduling (DLS) is a dynamic scheduling which uses dynamic level to assign a task into processing element. Dynamic level is the difference between static b-level and t-level. In each step, dynamic level is updated. The node with largest dynamic level with respect to processing element is scheduled first. In scheduling, it is very difficult to assign tasks to the processing elements.

In this thesis, we proposed efficient scheduling heuristics for skew data set and a new mode of heuristic i.e. immediate mode to solve the problems mentioned above. We also proposed three new heuristics for immediate mode scheduling.

Chapter 2

Basic Concepts

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

2.1 Task

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

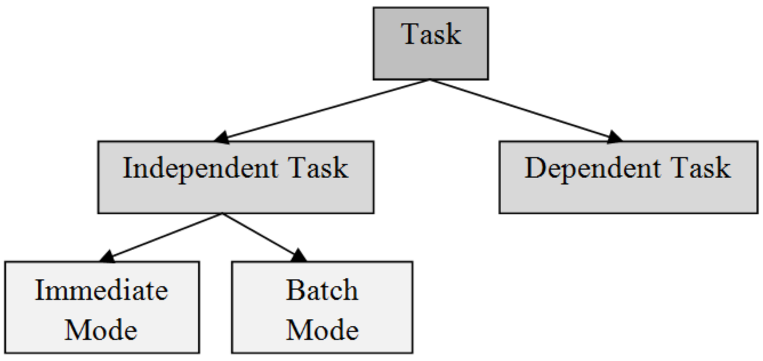

Figure 2.1: Hierarchy of task

2.1.1 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 effect 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.

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.

2.1.2 Dependent Task

Dependent task has a relationships 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 effect the computations. Alternatively, the tasks are scheduled in only one ways: T_i followed by T_j. Dependent tasks are represented in directed acyclic graph form or task graph form.

2.2 Machine

Machine is the producer or service in 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 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.

2.3 Types of Matrices

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

2.3.1 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).

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

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

2.3.2 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

2.3.3 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).

$$\begin{gathered}\text{Assume that, } E_{1,1} < E_{1,2} < ... < E_{1,n-1} < E_{1,n} \\ \text{then } \forall_i (E_{i,j} < E_{i,j+k} < ... < E_{i,j+k1} < E_{i,j+kx}) \text{ are true.} \\ \text{where } 1 \leq j \leq m,\ i = \text{any task } T_i \text{ ranges from 1 to } m, \\ j < j+k < j+k1 < j+k2 < ... < j+kx, \\ k < k1 < k2 < ... < kx\end{gathered}$$

Chapter 3

Related Work

In recent years, many algorithms have been designed to schedule the task efficiently in grid environment. As we know, task scheduling is a NP problem; it is difficult to find an optimal solution. Etminani et al., Liu et al. and Parsa et al. proposed a new algorithm based on two existing algorithm Min-Min and Max-Min [2] [7]. It chooses the two existing algorithm based on standard deviation of the expected CT [2]. Rasooli et al. introduced a rule based scheduling which contains two rules for resource selection and three rules for job queue [4]. Parsa et al. designed a task scheduling algorithm called RASA which selects Min-Min strategy to execute small task first and selects Max-Min strategy to execute large task first [7]. It seems to no starvation to the tasks.

Xiaoshan et al. proposed a Min-Min Heuristic. It is based on adaptive scheduling heuristics which includes Quality of Service guidance [1]. Sun et al. developed a priority based task scheduling (P-TSA). Tasks are sorted based on the priority and assign to processor by comparing P-TSA with existing grid scheduling algorithms. Zhang et al. introduces a new measurement called effective aggregated computing power (EACP) that strongly improves the performance of schedulers. Navimipour et al. used genetic algorithm (mutation and new approach of crossover) in linear genetic representation to overcome demerits of previous methods. Mansouri et al., Tang et al. and Abdi et al. proposed combine scheduling strategy in which several replication strategies and their performance is evaluated [13]. In order to improve data access efficiencies, the replica manager is used. For replica selection or deletion, this strategy considered bandwidth between the regions [13].

In recent years, many grid task scheduling have been designed to schedule task in parallel fashion. It is very difficult to find the solution which is applicable to all situations in grid environment. Senthilkumar et al. and Mehta et al. proposed a robust task scheduling for heterogeneous computing system. In this algorithm, each task arrival times and order of the task are not decided previously [11].

Many researchers have been proposed several algorithms on the parallel task scheduling. Each of them produces an optimal or semi-optimal schedule under various criteria. Ahmad et al. proposed a new algorithm for parallel task scheduling based on task duplication [14]. Due to duplication of tasks, it reduces the scheduling length. Liang et al. introduced a task scheduling using Breath First Search. It uses two queues: ready task queue (RTQ) and not ready task queue (NRTQ). By using these queues, it schedules the task in homogeneous environment. Jin et al. solves two problems in multiprocessor task scheduling: LU decomposition and Gauss-Jordan elimination. Dataflow in these problems are more frequent.

Real time has a time limit on computation. Yaashuwanth et al. and Liang et al. proposed a new scheduling algorithm based on real time system. Grid resource broker is responsible for allocation of a task to a particular resource which takes less time. It is also responsible for splitting the job into various tasks [3]. QoS Sufferage heuristic was proposed by Munir et al. [15]. Sufferage value is the difference between two best processors. It was proposed by Maheswaran et al. [16]. Maheswaran et al. also introduced one batch mode heuristic and two immediate mode heuristic. Sufferage heuristic is a batch mode heuristic. It schedules the tasks based on sufferage value. Two immediate mode heuristics are Swithing Algorithm (SA) and

K-Percent Best (KPB). SA uses MET and MCT in a cyclic manner to balance the load. KPB uses only a subset of machines.

3.1 Related Work on different Mode of Scheduling Heuristics

3.1.1 Minimum Execution Time

It assigns the task to the resource in First Come First Served (FCFS) basis. The resource which takes less Execution Time (ET) for a given task is scheduled first. Due to the above two reason, it leads to load imbalance. It may happen that a resource is completely idle. Let r indicates the number of resources and t indicates the number of tasks in a scenario. Then, O (r) time is required to assign a task to the resource [2]. Figure 3.1 shows the Gantt chart for MET (Table 3.1 data set). It gives a Makespan of 63 seconds.

3.1.2 Minimum Completion Time

Like MET, it assigns the task to the resource in FCFS basis. The resource which takes less Completion Time (CT) for a given task is scheduled first. Sometimes, the resource is busy while assigning the task. So, we must consider ready time of the resource. CT can be calculated using Equation 1. It may leads to load imbalance problem because of FCFS nature. Time complexity remains same as previous algorithm. Figure 3.2 shows the Gantt chart for MCT. It gives a Makespan of 55 seconds.

$$CT = ET + Ready\ \ time \tag{3.1}$$

3.1.3 Min-min

This algorithm does not follow FCFS sequence. It contains two criteria: MET and MCT. Minimum ET tasks are preferred before the maximum ET tasks. It is useful when few number of maximum ET tasks are present in Grid environment. The concept is choosing the task which holds minimum ET and assigns it to the resource which gives minimum CT. It may cause load imbalance problem if more number of larger tasks are present. It also causes starvation to maximum ET tasks. This algorithm takes O (t^2r) time [2]. Figure 3.3 shows the Gantt chart for Min-Min. It gives a Makespan of 53 seconds.

3.1.4 Max-min

Like Min-Min, it does not follow FCFS sequence. It is also combining MET and MCT concept. But, instead of MET, it takes maximum execution time. So, it prefers the maximum ET tasks before the minimum ET tasks. It gives a better schedule if few numbers of minimum ET tasks are present. Max-Min goal is to choose the task which has maximum ET and assigns it to the resource which gives minimum CT. It causes starvation to minimum ET tasks. It requires O (t^2r) time [2]. Figure 3.4 shows the Gantt chart for Max-Min. It gives a Makespan of 58 seconds.

3.1.5 Sufferage

In sufferage algorithm for each task, we have to find two best earliest completion time resources. The difference between the second and first resource completion time is called Sufferage Value (SV) [16]. For each task, SV is calculated. The task which suffers more or more SV value is assigned first. Time complexity is O(t^2r) [2]. Figure 3.5 shows the Gantt chart for Sufferage algorithm. It gives a Makespan of 40 seconds. In comparison to all scheduling algorithm, Sufferage gives better Makespan in Table 3.1 data set.

Table 3.1: 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

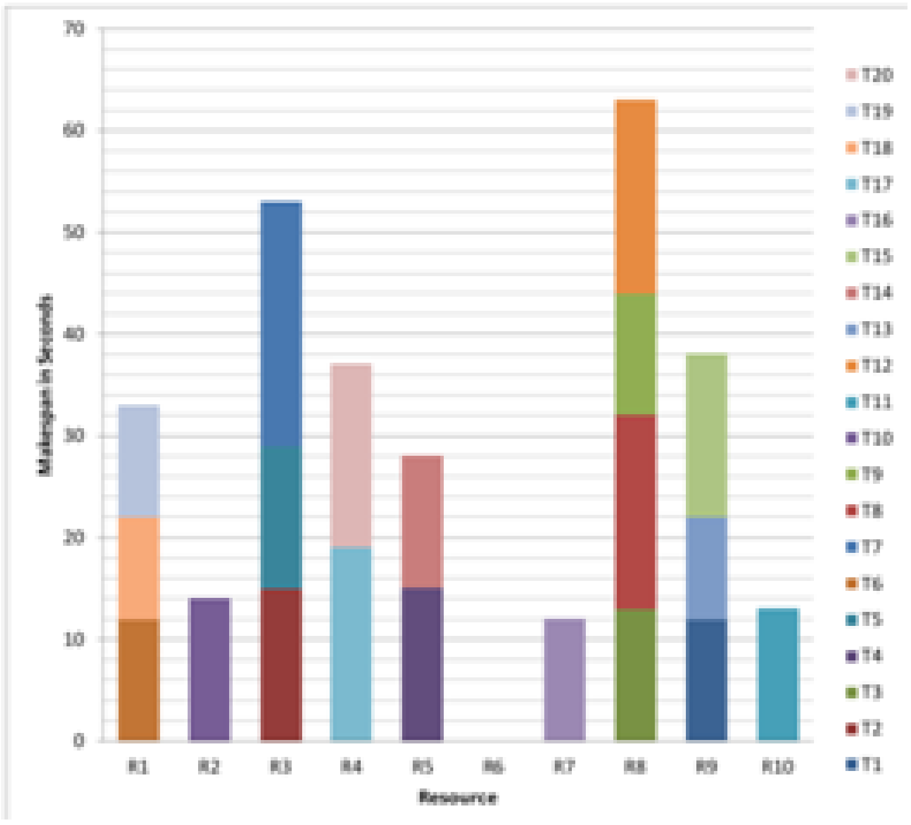

Figure 3.1: Gantt chart for MET

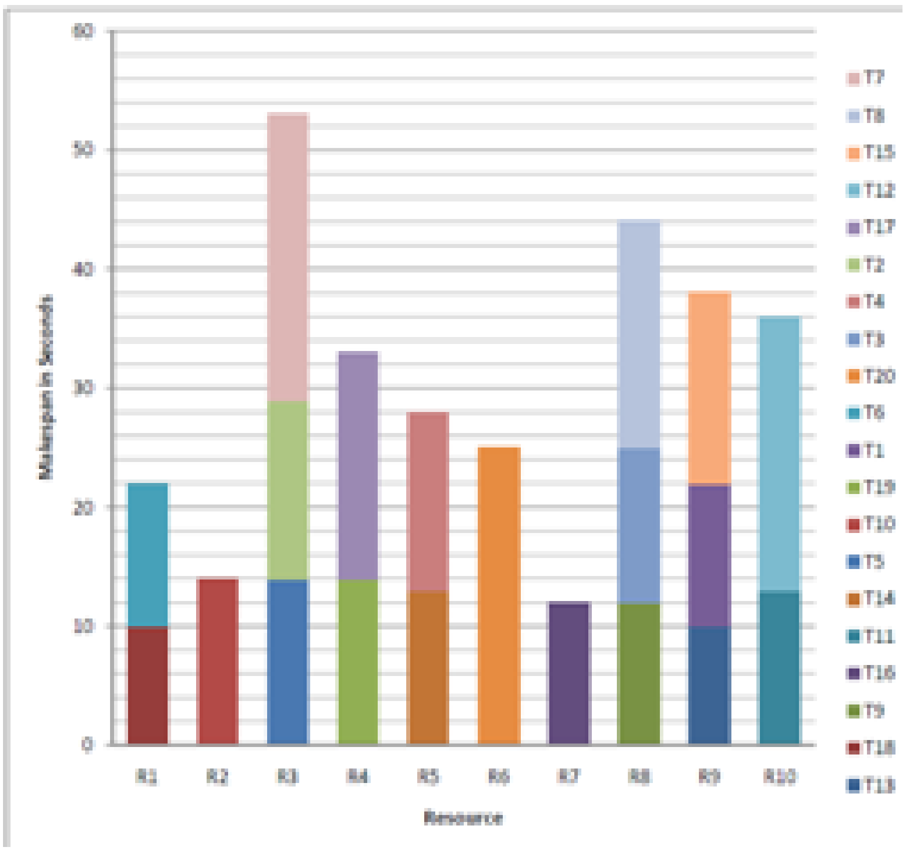

Figure 3.2: Gantt chart for MCT

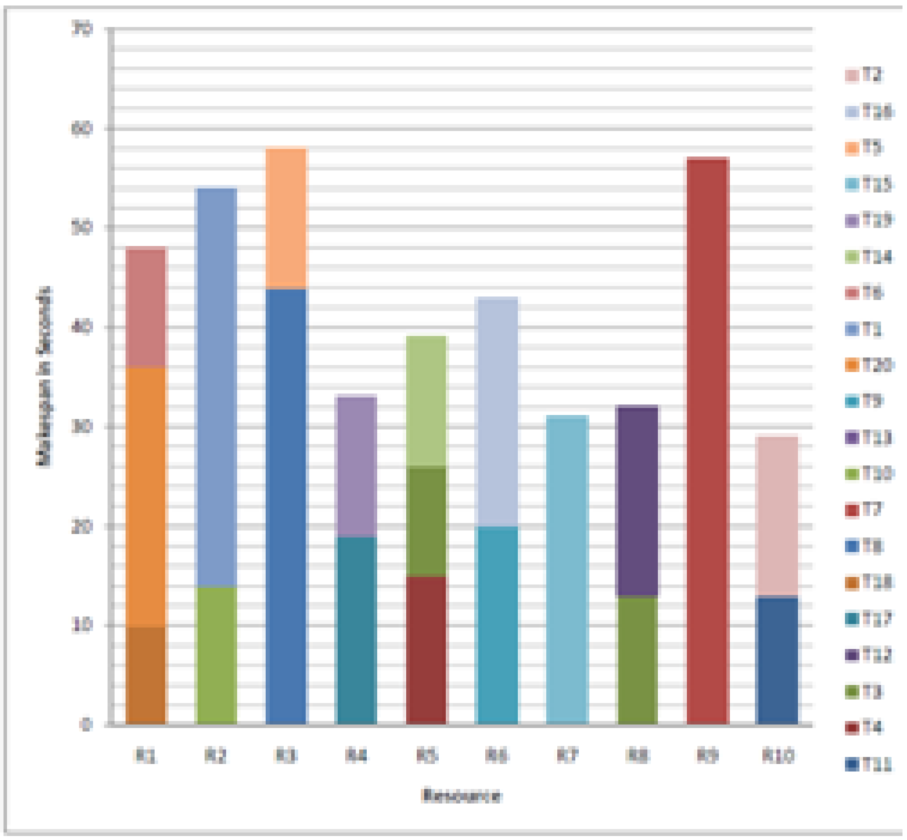

Figure 3.3: Gantt chart for Min-min

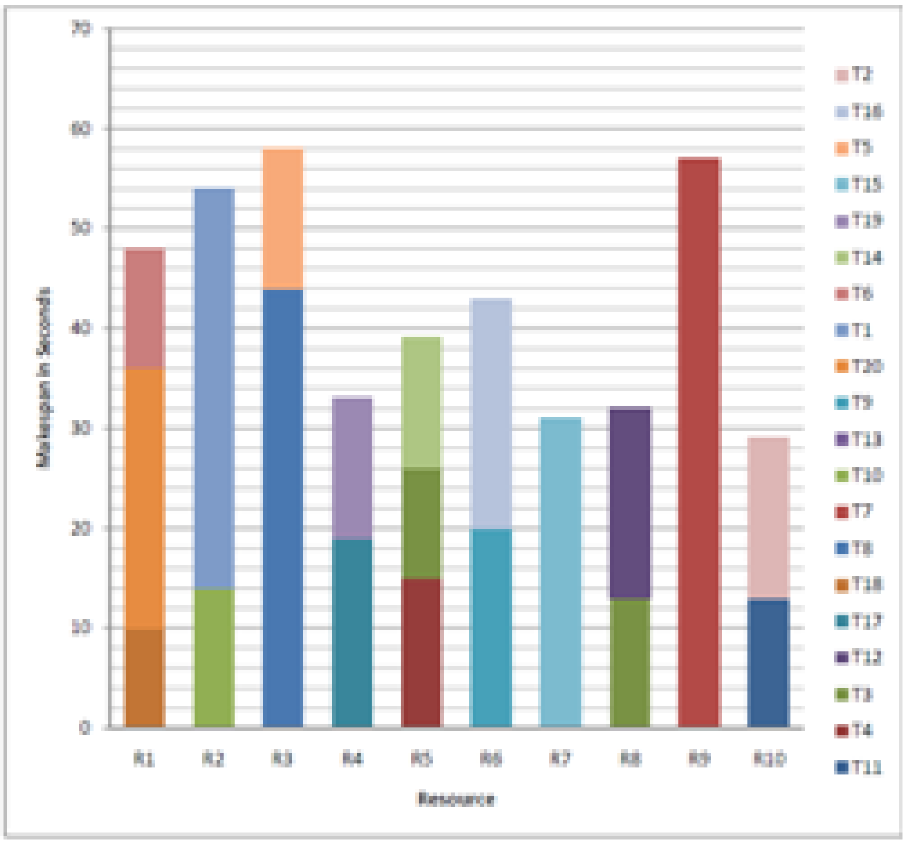

Figure 3.4: Gantt chart for Max-min

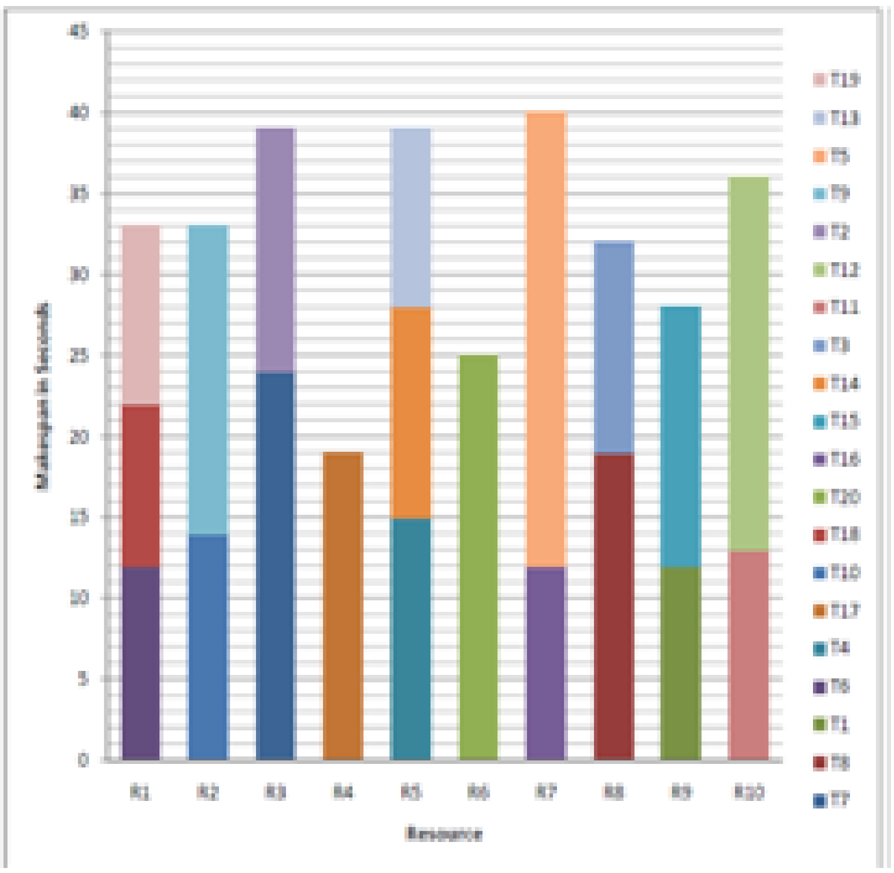

Figure 3.5: Gantt chart for Sufferage

3.1.6 Opportunistic Load Balancing

It assigns a task to the machine that becomes idle next. It is not taking execution time of the task and completion time of the task in to consideration. This heuristic requires $O(n)$ time to assign each task to the machine [2] [4].
Merits: It is very simple and inexpensive.
Demerits: Execution time of the task is not considered.
It can be mathematically expressed as follows: Let us consider the RTM matrix shown in Equation (3.2). The task T_1 is assigned to the least ready time machine as shown in Equation (3.3). The EET of task T_1 can be calculated as shown in Equation (3.4).

$$(R_1 \quad R_2 \quad R_3 \quad ... \quad R_n) \tag{3.2}$$

$$T_1 \longrightarrow min(R_1, R_2, R_3, ..., R_n) \tag{3.3}$$

$$T_1 \longrightarrow E_{1,1} + R_1, E_{1,2} + R_2, E_{1,3} + R_3, ..., E_{1,n} + R_n \tag{3.4}$$

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

3.1.7 K - Percent Best

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 (3.5). 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 (3.6). 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 [2].

$$(n') = n \times (K/100) \tag{3.5}$$

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

$$Heuristic = \begin{cases} MET & if K = 100/n \\ MCT & if K = 100 \\ KPB & Otherwise \end{cases} \tag{3.6}$$

3.1.8 Switching Algorithm

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 (3.7). 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.

$$\pi = r_{min}/r_{max} \tag{3.7}$$

3.1.9 QoS Guided Min-Min

Min-Min do not have QoS concept. It is possible that a task is not able to execute on a processor. This new concept was implemented in QoS guided Min-Min [1]. It divided the set of tasks into high QoS and low QoS tasks. High QoS task assignment is done first. It has a better makespan in comparison to Min-Min.

3.1.10 QoS Sufferage Heuristic

Like QoS Guided Min-Min, it divides the tasks into high QoS and low QoS tasks. It uses sufferage value to assign tasks to a processor. It also checks the sufferage value of assigned task with unassigned task. If the assigned task value is less than the unassigned one then assigned task is removed from the processor. The unassigned task is inserted to that processor.

Chapter 4

Efficient Scheduling Heuristics in Computational Grids

4.1 An Efficient Skewness Based Heuristic for Task Scheduling in Computational Grid

4.1.1 Heuristic Description

In this section, we present a skewness based task scheduling heuristic. At the first step to last step, all the steps are repeated until no meta-tasks are present in the TQ. In the second and third step, the meta-tasks are assigned to all the resources to calculate the completion time of each task in each individual processor. Completion time can be calculated using the Equation 3.1. It is shown in the fourth step.

In the step seven and eight, MCT of each meta-tasks are determined. This step gives a one dimensional array. Skewness are calculated in step eleven, using a formula shown in Equation 4.1.

$$Skewness = \frac{Q_3 + Q_1 - 2Q_2}{Q_3 - Q_1} \tag{4.1}$$

where Q_1 = First Quartile, Q_2 = Second Quartile, Q_3 = Third Quartile

To calculate skewness, we need to find the median. Then, we divide the one dimensional array in to two halves using 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 twelve, it checks whether skewness is less then 0 or not. If it's value is is greater than or equal to 0 then it selects max-min strategy in the first iteration. Otherwise, it selects min-min strategy. It is shown in the step thirteen and fourteen. Finally, it deletes the executed meta-task from TQ and updates the TQ in step sixteen. 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.

Algorithm 1 - Skewness Based Heuristic

```
1: while $TQ$ $!= NULL$
2:     for all meta-tasks $T_i$ in $TQ$
3:         for all machines $M_j$
4:             $C_{i,j} = E_{i,j} + R_j$
5:         end for
6:     end for
7:     for all meta-tasks $T_i$ in $TQ$
8:         Find minimum $C_{i,j}$ and machine $M_j$ that holds it.
9:     end for
10:    Sort the meta-tasks in ascending order of $C_{i,j}$
11:    Calculate Skewness.
         Skewness = $\frac{Q_3+Q_1-2Q_2}{Q_3-Q_1}$
12:    If Skewness $\geq 0$
13:    then assign meta-task $T_i$ to machine $M_k$ that holds maximum $C_{i,k}$.
14:    else assign meta-task $T_i$ to machine $M_k$ that holds minimum $C_{i,k}$.
15:    end if
16:    Delete the meta-task $T_i$ and update $TQ$.
17: end while
18: Calculate Makespan and $AMU$.
```

4.2 Intermediate Mode

By considering the merits and demerits of immediate mode and batch mode, we have proposed intermediate mode heuristic. Intermediate mode heuristic is a variation of

batch mode heuristic. The value of ζ (number of tasks arrival) varies from 2 to α-1. If $\zeta = 1$, intermediate mode heuristic acts like immediate mode heuristic. If $\zeta = \alpha$, it acts like batch mode heuristic. Note that, α is unknown in intermediate mode.
In this mode, we have taken a random function to determine the ζ value. In each iteration, ζ value is determined. Based on ζ value, numbers of tasks are computed. For 512 $\times$ 16 instances, the values of ζ are 58, 46, 62, 38, 40, 36, 36, 60, 51, 50 and 35. It means 58 tasks are executed in first iteration, 46 tasks are executed in second iteration and so on.
we have taken τ value as 32 to 64 for simplicity. In real life situation, it may vary. All the instances e.g. 512 $\times$ 16, 1024 $\times$ 32, 2048 $\times$ 64 has τ value as 32 to 64.

4.3 Implementation and results

4.3.1 Skewness Based Heuristic

Table 4.1: Makespan Values for Min-Min, Max-Min and Skewness heuristic

Instance	**Min-Min**	**Max-Min**	**Skewness**
50 × 5	1.2023E+04	9.8310E+03	9.8240E+03
50 × 10	1.0510E+04	9.3010E+03	9.3010E+03
50 × 15	1.0201E+04	9.3000E+03	9.3000E+03
100 × 5	1.9045E+04	1.4085E+04	1.4081E+04
100 × 10	1.2021E+04	9.3030E+03	9.3030E+03
100 × 15	1.0331E+04	9.1900E+03	9.1900E+03
1000 × 5	7.8848E+04	8.1964E+04	8.0117E+04
1000 × 10	4.1502E+04	4.0530E+04	3.9305E+04
1000 × 15	2.9294E+04	2.6468E+04	2.5612E+04
10000 × 5	7.1516E+05	7.0218E+05	6.8232E+05
10000 × 10	4.0632E+05	3.6500E+05	3.5322E+05
10000 × 15	7.5693E+05	5.5530E+05	5.5530E+05

Table 4.2: Resource Utilisation Values for Min-Min, Max-Min and Skewness Heuristic

Instance	**Min-Min**	**Max-Min**	**Skewness**
50 × 5	0.7999	0.9838	0.9813
50 × 10	0.4382	0.4977	0.4970
50 × 15	0.2355	0.2593	0.2593
100 × 5	0.7263	0.9910	0.9907
100 × 10	0.4916	0.6416	0.6410
100 × 15	0.1809	0.2068	0.2067
1000 × 5	0.9596	0.9983	0.9993
1000 × 10	0.8826	0.9976	0.9949
1000 × 15	0.8121	0.9953	0.9954
10000 × 5	0.8891	0.9998	0.9999
10000 × 10	0.7993	0.9996	0.9994
10000 × 15	0.3310	0.5014	0.4873

4.3.2 Intermediate mode heuristics

Makespan and RU of min-min heuristic in intermediate mode are shown in Table 4.3 and Table 4.4 respectively. Makespan and RU of max-min heuristic in intermediate mode are shown in Table 4.5 and Table 4.6 respectively. Each instances are computed under 512 $\times$ 16 (Case 1), 1024 $\times$ 32 (Case 2) and 2048 $\times$ 64 (Case 3). Min-min graphical representation of makespan value and RU value for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo are shown in Figure 4.1 and Figure 4.3 respectively. Max-min graphical representation of makespan value and RU value for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo are shown in Figure 4.2 and Figure 4.4 respectively. Here, t indicates type of consistency e.g. consistent, inconsistent and semi-consistent.

Table 4.3: Makespan Values for Min-Min Heuristic in Intermediate Mode

Instance	**Case 1**	**Case 2**	**Case 3**
u_c_hihi	1.0166E+07	2.8030E+07	2.5465E+07
u_c_hilo	1.7060E+05	2.8572E+06	2.4457E+06
u_c_lohi	3.2393E+05	2.7269E+03	2.5476E+03
u_c_lolo	5.8218E+03	2.8531E+02	2.4985E+02
u_i_hihi	3.9365E+06	7.0648E+06	3.5620E+06
u_i_hilo	8.5327E+04	6.6892E+05	3.9054E+05
u_i_lohi	1.3295E+05	7.2203E+02	3.6930E+02
u_i_lolo	2.9201E+03	6.9890E+01	4.0690E+01
u_s_hihi	5.5299E+06	1.7980E+07	1.5191E+07
u_s_hilo	1.1047E+05	1.6746E+06	1.3569E+06
u_s_lohi	1.7105E+05	1.7268E+03	1.4208E+03
u_s_lolo	4.0299E+03	1.7670E+02	1.5070E+02

Table 4.4: Resource Utilisation Values for Min-Min Heuristic in Intermediate Mode

Instance	**Case 1**	**Case 2**	**Case 3**
u_c_hihi	0.9256	0.9105	0.9150
u_c_hilo	0.9541	0.9200	0.9274
u_c_lohi	0.9324	0.9007	0.9155
u_c_lolo	0.9384	0.9130	0.9334
u_i_hihi	0.9114	0.9041	0.8763
u_i_hilo	0.9645	0.8993	0.8659
u_i_lohi	0.9222	0.8878	0.9203
u_i_lolo	0.9617	0.8914	0.8260
u_s_hihi	0.9309	0.8910	0.8634
u_s_hilo	0.9453	0.9003	0.8907
u_s_lohi	0.8860	0.8641	0.8707
u_s_lolo	0.9438	0.8492	0.8588

Table 4.5: Makespan Values for Max-Min Heuristic in Intermediate Mode

Instance	**Case 1**	**Case 2**	**Case 3**
u_c_hihi	1.2800E+07	3.5985E+07	3.1587E+07
u_c_hilo	2.0339E+05	3.6091E+06	3.0689E+06
u_c_lohi	4.1913E+05	3.5161E+03	3.2692E+03
u_c_lolo	6.8891E+03	3.7003E+02	3.0880E+02
u_i_hihi	5.5328E+06	8.8184E+06	4.0519E+06
u_i_hilo	1.1861E+05	8.1415E+05	4.3581E+05
u_i_lohi	1.8903E+05	8.5709E+02	4.1651E+02
u_i_lolo	3.9876E+03	8.3320E+01	4.4050E+01
u_s_hihi	8.1120E+06	2.1557E+07	1.7156E+07
u_s_hilo	1.4574E+05	1.9977E+06	1.5633E+06
u_s_lohi	2.3635E+05	2.0091E+03	1.6200E+03
u_s_lolo	5.3405E+03	2.2031E+02	1.7633E+02

Table 4.6: Resource Utilisation Values for Max-Min Heuristic in Intermediate Mode

Instance	**Case 1**	**Case 2**	**Case 3**
u_c_hihi	0.9962	0.9736	0.9619
u_c_hilo	0.9941	0.9763	0.9630
u_c_lohi	0.9905	0.9560	0.9696
u_c_lolo	0.9900	0.9820	0.9772
u_i_hihi	0.9747	0.9523	0.8540
u_i_hilo	0.9956	0.9193	0.8977
u_i_lohi	0.9682	0.9551	0.9062
u_i_lolo	0.9788	0.9429	0.8610
u_s_hihi	0.9397	0.9684	0.9265
u_s_hilo	0.9848	0.9591	0.9403
u_s_lohi	0.9863	0.9598	0.9393
u_s_lolo	0.9799	0.9349	0.9141

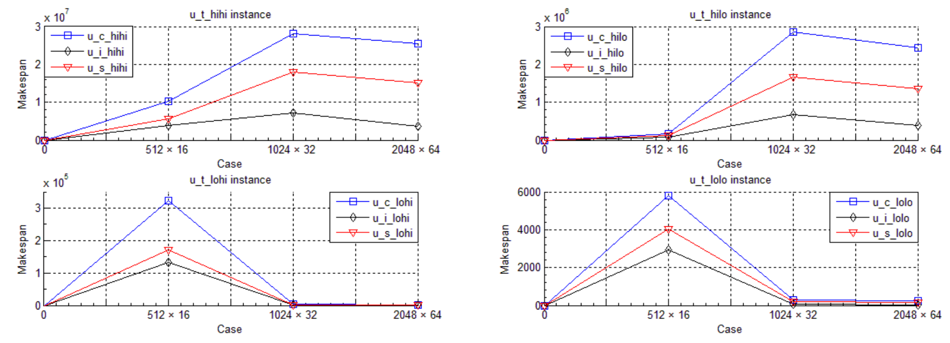

Figure 4.1: Min-min graphical representation of makespan value for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo instances

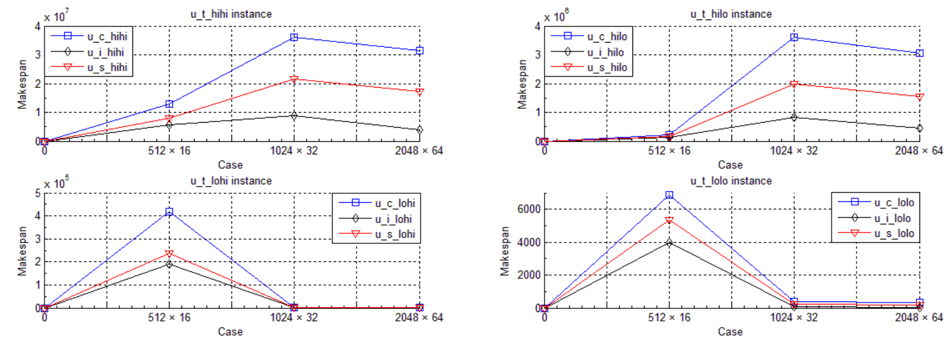

Figure 4.2: Max-min graphical representation of makespan value for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo instances

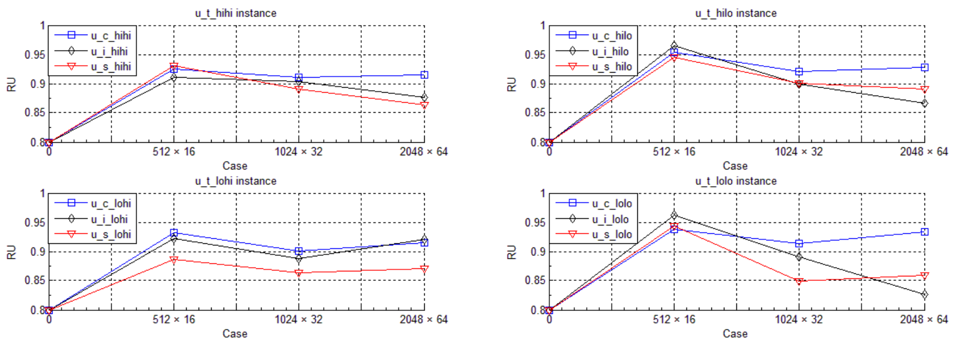

Figure 4.3: Min-min graphical representation of RU for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo instances)

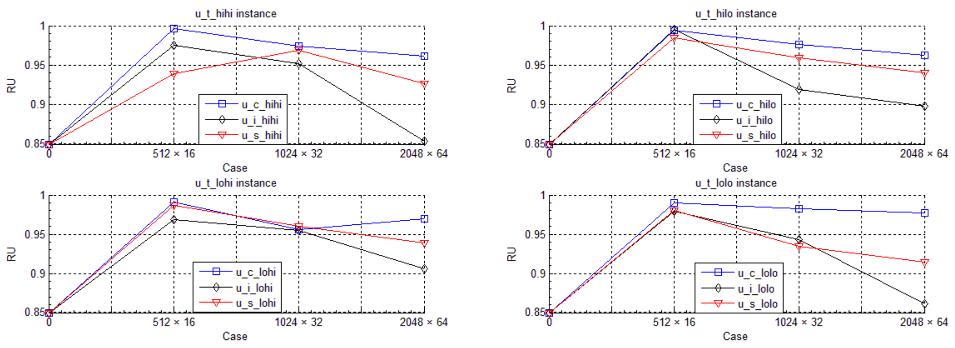

Figure 4.4: Max-min graphical representation of RU for u_t_hihi, u_t_hilo, u_t_lohi and u_t_lolo instances

Chapter 5

X-DualMake Heuristics for Independent Tasks in Computational Grids

We propose three immediate mode heuristics: First-DualMake (F-DM) heuristic, Best-DualMake (BDM) heuristic, and Worst-DualMake (WDM) heuristic. It is based on the idle time of the resource. The goal of these heuristic is to reduce the idle time of each resource instead of task completion time. For this, we need to calculate the resource ready time. It is called as Pre-Makespan. It is the maximum ready time of all available resources. After each task has been executed, the Makespan is recalculated. It is called as Post-Makespan. In each heuristic, the name DualMake stands for Pre-Makespan and Post-Makespan.

5.1 F-DM Heuristic

In this heuristic, the first indicates the first available resource for the upcoming task. If no resource is available, then the first task is assigned to most probable idle time resource. It searches for first available resources which has the enough idle time. It stops when it finds an available resource. Makespan is calculated after each assignment of task.

Algorithm 2 - F-DM

```
1: for all resource $R_j$ // $R_j$ = Resource j
2: Read $RT_j$ // $RT_j$ = Ready Time of Resource j
3: end for
4: Calculate the $M_{pr}$ // $M_{pr}$ = Pre-Makespan
5: for all resource $R_j$
6: $T_I[R_j]$ = $M_{pr}[R_j]$ - $RT[R_j]$ // $T_I[R_j]$ = Idle Time of Resource j, $M_{pr}[R_j]$ = Pre-Makespan of Resource j,
$RT[R_j]$ = Ready Time of Resource j
7: end for
8: Sort the resource $R_j$ with respect to $T_I[R_j]$ in ascending order
9: do task $T_i$
10: for all resource $R_j$
11: $T_{ij} = T_I[R_j] - ET_{ij}$ // $ET_{ij}$ = Execution Time of task $T_i$ on Resource $R_j$
12: $R_c = R_j$ // $R_c$ = Current Resource
13: if $T_{ij} \geq 0$ // $T_{ij}$ = Remaining Idle Time of Resource j if $T_i$ assigns to $R_j$
14: Go to Step 18
15: end if
16: end for
17: end do
18: Assign $T_i$ to resource $R_c$
19: Calculate the $M_{po}$ // $M_{po}$ = Post-Makespan
```

5.2 B-DM Heuristic

In this heuristic, the best indicates the best available resource for the upcoming task. If no resource is available to map, then the task is assigned to most probable idle time resource. It searches the entire available resources and chooses a resource which is the smallest idle time. Unlike the F-DM, this heuristic is an alternative to choose one resource. It works like the F-DM if no resource has sufficient idle time.

Algorithm 3 - B-DM

```
1: for all resource $R_j$ // $R_j$ = Resource j
2: Read $RT_j$ // $RT_j$ = Ready Time of Resource j
3: end for
4: Calculate the $M_{pr}$ // $M_{pr}$ = Pre-Makespan
5: for all resource $R_j$
6: $T_I[R_j]$ = $M_{pr}[R_j]$ - $RT[R_j]$ // $T_I[R_j]$ = Idle Time of Resource j, $M_{pr}[R_j]$ = Pre-Makespan of Resource j,
$RT[R_j]$ = Ready Time of Resource j
7: end for
8: do task $T_i$
9: for all resource $R_j$
10: $T_{ij}$ = $T_I[R_j]$ - $ET_{ij}$ // $ET_{ij}$ = Execution Time of task $T_i$ on Resource $R_j$
11: end for
12: Sort $T_{ij}$ and its corresponding $R_j$ in ascending order
13: do until $T_{ij} \leq 0$ && j $\leq$ Y
14: j = j + 1
15: end do
16: end do
17: Assign $T_i$ to resource in index j
18: Calculate the $M_{po}$ // $M_{po}$ = Post-Makespan
```

5.3 W-DM Heuristic

upcoming task. It is similar to B-DM. But, it selects the worst resource instead of best resource. Sometimes, W-DM outperforms than F-DM and B-DM. It searches the entire available resources and chooses a resource which is the largest idle time. It assigns the task to the resource which holds the largest idle time. It is same as MCT heuristic. The idle time of a task in W-DM same as the earliest completion time of a task. As a part of the complete idle time scenario, we have shown it.

Algorithm 4 - W-DM

```
1: for all resource $R_j$ // $R_j$ = Resource j
2: Read $RT_j$ // $RT_j$ = Ready Time of Resource j
3: end for
4: Calculate the $M_{pr}$ // $M_{pr}$ = Pre-Makespan
5: for all resource $R_j$
6: $T_I[R_j]$ = $M_{pr}[R_j]$ - $RT[R_j]$ // $T_I[R_j]$ = Idle Time of Resource j, $M_{pr}[R_j]$ = Pre-Makespan of Resource j,
$RT[R_j]$ = Ready Time of Resource j
7: end for
8: do task $T_i$
9: for all resource $R_j$
10: $T_{ij}$ = $T_I[R_j]$ - $ET_{ij}$ // $ET_{ij}$ = Execution Time of task $T_i$ on Resource $R_j$
11: end for
12: Find max($T_{ij}$)
13: end do
14: Assign $T_i$ to resource $R_j$
15: Calculate the $M_{po}$ // $M_{po}$ = Post-Makespan
```

5.4 Implementation and results

The proposed heuristics are implemented and compared using the instances by Braun et al. [10]. We have taken different sizes of ETC matrices such as 512 tasks and 16 resources and 1024 tasks and 32 resources. In each size, 12 different types of matrices are compared. The instances are consisting of three parameters: distribution, the nature of the matrix and task-resource heterogeneity. The general representations of these instances are u_a_bbcc.0. Here, u indicate the distribution is uniform Followed by, a indicates the nature of the matrix i.e. c-consistent, i-inconsistent and s-semi-consistent. Next, bb indicates the task heterogeneity i.e. either hi-high or lo-low, and cc indicates the resource heterogeneity i.e. either hi-high or lo-low. In each nature of a matrix, we have four combinations (hihi, hilo, lohi, and lolo). We have taken k = 20% in KPB heuristic and l = 0.6 and h = 0.9 in SA heuristic for all types of instance. First, we simulate for 512 tasks and 16 resources. The Makespan comparisons of consistent, inconsistent and semi-consistent metrics are shown in Figure 5.1, Figure 5.2, and Figure 5.3 respectively. The resource utilization comparison of consistent, inconsistent and semi-consistent metrics are shown in Figure 5.4.

Table 5.1: Makespan Values for MET, MCT, OLB, KPB, SA, F-DM, B-DM AND W-DM Heuristics using 512 tasks and 16 resources

Instance	**MET**	**MCT**	**OLB**	**KPB**	**SA**	**F-DM**	**B-DM**	**W-DM**
u_c_hihi	4.7472E+07	1.1423E+07	1.4377E+07	2.2972E+07	1.2613E+07	1.3359E+07	1.1980E+07	1.1423E+07
u_c_hilo	1.1851E+06	1.8589E+05	2.2105E+05	4.8110E+05	1.9455E+05	1.9837E+05	1.9480E+05	1.8589E+05
u_c_lohi	1.4531E+06	3.7830E+05	4.7736E+05	7.1483E+05	4.2627E+05	4.4870E+05	3.9477E+05	3.7830E+05
u_c_lolo	3.9582E+04	6.3601E+03	7.3066E+03	1.6120E+04	8.1671E+03	6.7718E+03	6.4801E+03	6.3601E+03
u_i_hihi	4.5085E+06	4.4136E+06	2.6102E+07	4.2098E+06	4.6922E+06	1.0856E+07	7.6376E+06	4.4136E+06
u_i_hilo	9.6610E+04	9.4856E+04	2.7279E+05	8.9698E+04	1.0298E+05	1.8464E+05	1.3080E+05	9.4856E+04
u_i_lohi	1.8569E+05	1.4382E+05	8.3361E+05	1.3682E+05	1.4391E+05	3.3721E+05	2.5974E+05	1.4382E+05
u_i_lolo	3.3993E+03	3.1374E+03	8.9380E+03	3.0111E+03	3.4853E+03	5.4797E+03	4.4006E+03	3.1374E+03
u_s_hihi	2.5162E+07	6.6939E+06	1.9465E+07	6.9423E+06	7.1277E+06	1.2331E+07	9.3984E+06	6.6939E+06
u_s_hilo	6.0536E+05	1.2659E+05	2.5036E+05	1.4034E+05	1.4905E+05	1.6985E+05	1.5567E+05	1.2659E+05
u_s_lohi	6.7469E+05	1.8615E+05	6.0323E+05	1.9995E+05	1.9432E+05	3.6149E+05	2.8610E+05	1.8615E+05
u_s_lolo	2.1042E+04	4.4361E+03	8.9384E+03	5.0071E+03	5.8370E+03	6.2136E+03	5.5996E+03	4.4361E+03

Table 5.2: Resource Utilisation Values for MET, MCT, OLB, KPB, SA, F-DM, B-DM AND W-DM Heuristics using 512 tasks and 16 resources

Instance	**MET**	**MCT**	**OLB**	**KPB**	**SA**	**F-DM**	**B-DM**	**W-DM**
u_c_hihi	1	0.9539	0.9467	0.9948	0.8905	0.9173	0.9740	0.9539
u_c_hilo	1	0.9707	0.9203	0.9992	0.9209	0.9681	0.9736	0.9707
u_c_lohi	1	0.9690	0.9285	0.9956	0.8326	0.9206	0.9762	0.9690
u_c_lolo	1	0.9515	0.9232	0.9984	0.7279	0.9566	0.9733	0.9515
u_i_hihi	0.6286	0.9329	0.9512	0.9438	0.8469	0.9546	0.9801	0.9329
u_i_hilo	0.7506	0.9598	0.9559	0.9412	0.8167	0.9196	0.9840	0.9598
u_i_lohi	0.5366	0.9496	0.9340	0.9464	0.9481	0.9547	0.9604	0.9496
u_i_lolo	0.7404	0.9657	0.9796	0.9688	0.7977	0.9674	0.9782	0.9657
u_s_hihi	0.1971	0.9283	0.9671	0.9326	0.8631	0.9863	0.9670	0.9283
u_s_hilo	0.2142	0.9383	0.9246	0.9507	0.7813	0.9802	0.9923	0.9383
u_s_lohi	0.2167	0.9539	0.9620	0.9656	0.8911	0.9831	0.9813	0.9539
u_s_lolo	0.2212	0.9519	0.9510	0.9628	0.7086	0.9514	0.9758	0.9519

Table 5.3: Makespan Values for MET, MCT, OLB, KPB, SA, F-DM, B-DM AND W-DM Heuristics using 1024 tasks and 32 resources

Instance	**MET**	**MCT**	**OLB**	**KPB**	**SA**	**F-DM**	**B-DM**	**W-DM**
u_c_hihi	1.5447E+08	3.2833E+07	4.2817E+07	5.3605E+07	3.7301E+07	3.5554E+07	3.3651E+07	3.2833E+07
u_c_hilo	1.5504E+07	3.2458E+06	4.4054E+06	5.4320E+06	3.2458E+06	3.9253E+06	3.9253E+06	3.2458E+06
u_c_lohi	1.4151E+04	3.0587E+03	4.4132E+03	4.8931E+03	3.0587E+03	3.7370E+03	3.2375E+03	3.0587E+03
u_c_lolo	1.5675E+03	3.2628E+02	4.4475E+02	5.4268E+02	4.1438E+02	4.0760E+02	3.3570E+02	3.2628E+02
u_i_hihi	7.4620E+06	7.5671E+06	8.4914E+07	7.4932E+06	7.5671E+06	2.3937E+07	1.6626E+07	7.5671E+06
u_i_hilo	7.6598E+05	7.1313E+05	7.8322E+06	6.9501E+05	7.1313E+05	2.7569E+06	1.5649E+06	7.1313E+05
u_i_lohi	8.5439E+02	7.5410E+02	8.6143E+03	7.2986E+02	7.5410E+02	2.2689E+03	1.7735E+03	7.5410E+02
u_i_lolo	9.1120E+01	7.2390E+01	9.0081E+02	7.1610E+01	7.2390E+01	2.8137E+02	1.5366E+02	7.2390E+01
u_s_hihi	8.4821E+07	1.9008E+07	7.7562E+07	2.8102E+07	1.9008E+07	3.5030E+07	2.6025E+07	1.9008E+07
u_s_hilo	8.0988E+06	1.8255E+06	8.1962E+06	2.6997E+06	1.8255E+06	3.7373E+06	2.4675E+06	1.8255E+06
u_s_lohi	8.3377E+03	1.8220E+03	7.9978E+03	2.6423E+03	1.8220E+03	4.3091E+03	2.4676E+03	1.8220E+03
u_s_lolo	8.0161E+02	1.9423E+02	8.2890E+02	2.7351E+02	1.9423E+02	3.6628E+02	2.6774E+02	1.9423E+02

Table 5.4: Resource Utilisation Values for MET, MCT, OLB, KPB, SA, F-DM, B-DM AND W-DM Heuristics using 1024 tasks and 32 resources

Instance	**MET**	**MCT**	**OLB**	**KPB**	**SA**	**F-DM**	**B-DM**	**W-DM**
u_c_hihi	1	0.9355	0.8980	0.9957	0.8058	0.9702	0.9698	0.9355
u_c_hilo	1	0.9461	0.8886	0.9896	0.9461	0.8625	0.9562	0.9461
u_c_lohi	1	0.9226	0.8625	0.9832	0.9226	0.8623	0.9424	0.9226
u_c_lolo	1	0.9501	0.8646	0.9936	0.7075	0.8666	0.9713	0.9501
u_i_hihi	0.6605	0.9122	0.9410	0.9255	0.9122	0.9549	0.9665	0.9122
u_i_hilo	0.6058	0.9126	0.9621	0.9181	0.9126	0.9726	0.9607	0.9126
u_i_lohi	0.5799	0.9167	0.9613	0.9319	0.9167	0.9741	0.9733	0.9167
u_i_lolo	0.5264	0.9178	0.9302	0.9113	0.9178	0.9885	0.9768	0.9178
u_s_hihi	0.0577	0.9134	0.9290	0.3252	0.9134	0.9594	0.9711	0.9134
u_s_hilo	0.0602	0.9326	0.9510	0.3334	0.9326	0.9771	0.9746	0.9326
u_s_lohi	0.0604	0.8980	0.9430	0.3326	0.8980	0.8548	0.9705	0.8980
u_s_lolo	0.0614	0.9037	0.9489	0.3409	0.9037	0.8977	0.9711	0.9037

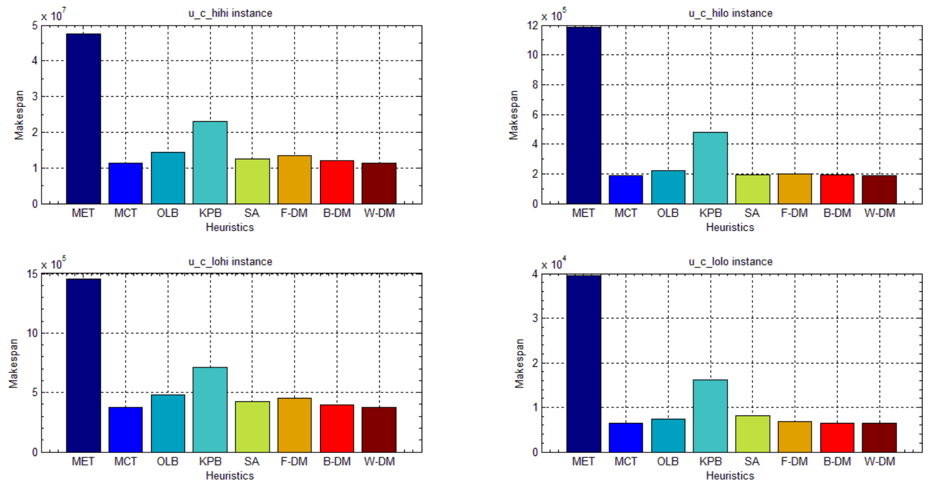

Figure 5.1: Makespan comparisons for u_c_bbcc instances (512 tasks and 16 resources)

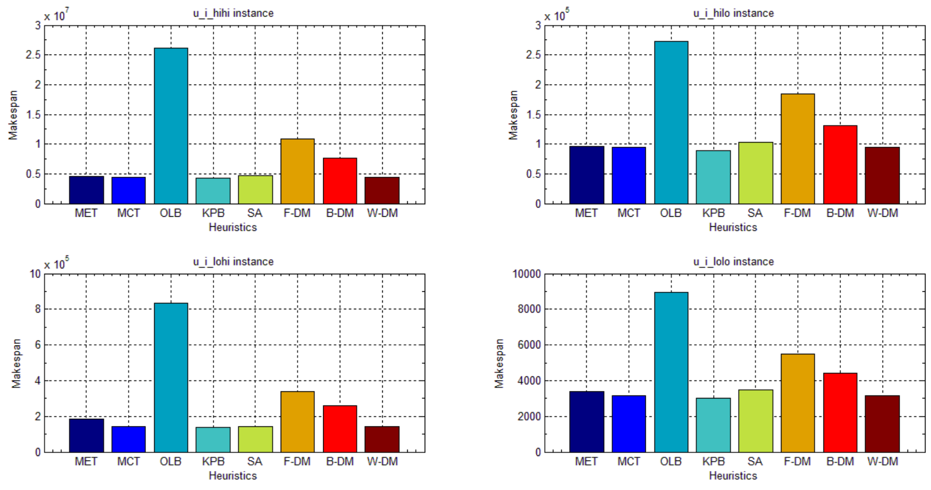

Figure 5.2: Makespan comparisons for u_i_bbcc instances (512 tasks and 16 resources)

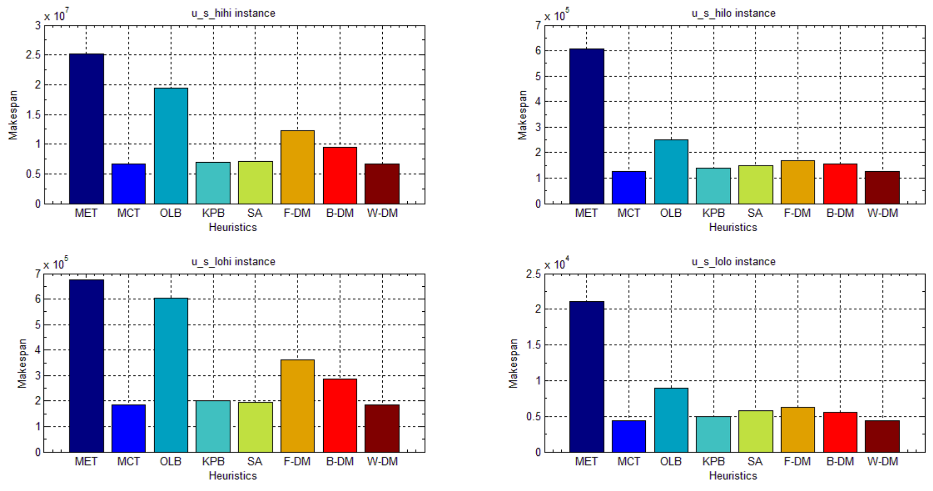

Figure 5.3: Makespan comparisons for u_s_bbcc instances (512 tasks and 16 resources)

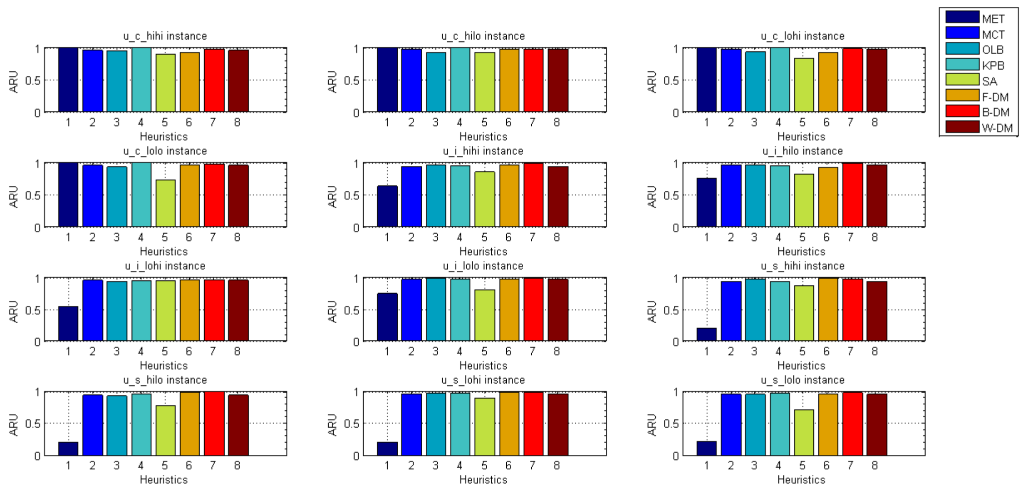

Figure 5.4: Average Resource Utilisation comparisons for u_a_bbcc instances (512 tasks and 16 resources)

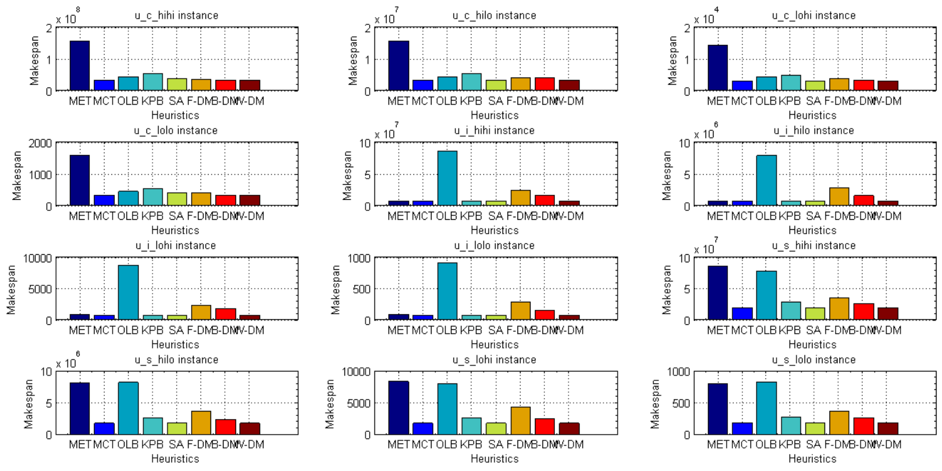

Figure 5.5: Makespan comparisons for u_a_bbcc instances (1024 tasks and 32 resources)

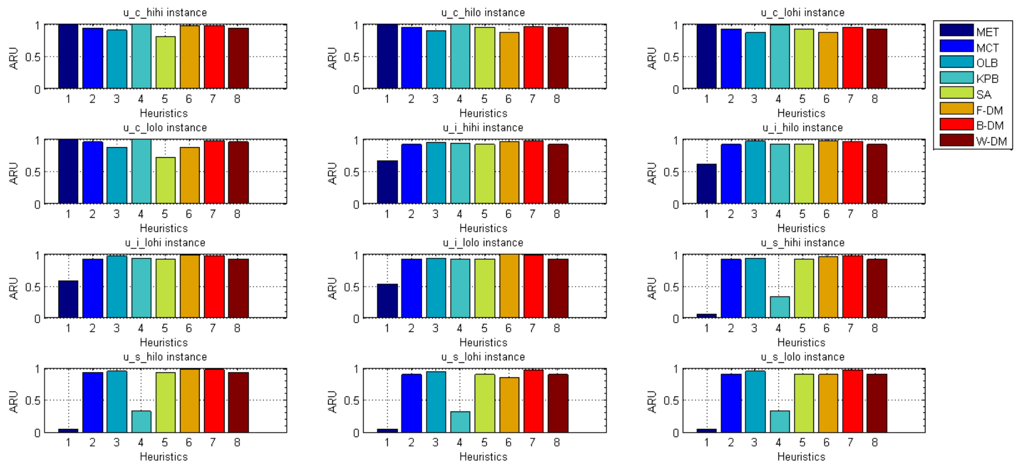

Figure 5.6: Average Resource Utilisation comparisons for u_a_bbcc instances (1024 tasks and 32 resources)

Chapter 6

Conclusion

In this thesis, eight immediate mode heuristics are discussed and implemented. No heuristic is giving better result in all instances. So, scheduling in heterogeneous grid environment is a NP-Complete problem. MCT heuristic is giving better results in consistent matrices and semiconsistent metrics. KPB is giving better results in an inconsistent scenario [9]. Among the three proposed heuristic, W-DM is similar to MCT. After MET, B-DM gives better results in consistent and semi-consistent scenario. We have taken two performance measures to compare each heuristic: Makespan and resource utilization. The proposed heuristics can be implemented in a real heterogeneous grid environment. The work can be extended by using decentralized task assignment, deadline for task assignment and fault-tolerance in grid resource broker. The grid resource broker is a centralized module. If the broker fails then it leads to failure of the grid environment. So, we need a fault-tolerance based broker. The work can also be extended using preemptive tasks assignment.

Dissemination

1 P. Agrawal, P. M. Khilar and D. P. Mohapatra, "Intermediate Mode Scheduling in Computational Grid", *3rd International Conference on Advances in Computing and Communications (ACC), IEEE*, 2013. (Accepted)

2 P. Agrawal, P. M. Khilar and D. P. Mohapatra, "X-DualMake: A Novel Immediate Mode Heuristics in Grid", *International Journal of Grid and Utility Computing, Inderscience*, 2013. (Communicated)

Bibliography

[1] XiaoShan He, XianHe Sun, and Gregor von Laszewski. Qos guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Technol.*, 18(4):442–451, July 2003.

[2] K. Etminani and M. Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1–7, 2007.

[3] M. Hemamalini. Article: Review on grid task scheduling in distributed heterogeneous environment. *International Journal of Computer Applications*, 40(2):24–30, February 2012. Published by Foundation of Computer Science, New York, USA.

[4] A. Rasooli, M. Mirza-Aghatabar, and S. Khorsandi. Introduction of novel rule based algorithms for scheduling in grid computing systems. In *Modeling Simulation, 2008. AICMS 08. Second Asia International Conference on*, pages 138–143, 2008.

[5] A. Chakrabarti. *Grid Computing Security.* Springer-Verlag Berlin Heidelberg.

[6] Dingju Zhu and Jianping Fan. Aggregation grid. In *Integration Technology, 2007. ICIT '07. IEEE International Conference on*, pages 357–364, 2007.

[7] Saeed Parsa and Reza Entezari-Maleki. Rasa: A new task scheduling algorithm in grid environment. *World Applied sciences journal*, 7:152–160, 2009.

[8] R. Buyya. *High Performance Cluster Computing.* Pearson Education.

[9] Fang Dong, Junzhou Luo, Lisha Gao, and Liang Ge. A grid task scheduling algorithm based on qos priority grouping. In *Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference*, pages 58–61, 2006.

[10] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Blni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, and Richard F Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[11] B. SenthilKumar, P. Chitra, and G. Prakash. Robust task scheduling on heterogeneous computing systems using segmented maxr-minct. 1(2):63–65.

[12] T Kokilavani and DI George Amalarethinam. Load balanced min-min algorithm for static meta-task scheduling in grid computing. *International Journal of Computer Applications*, 20(2):43–49, 2011.

[13] N. Mansouri, G. Dastghaibyfard, and A. Horri. A novel job scheduling algorithm for improving data grid's performance. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, pages 142–147, 2011.

[14] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, pages 79–87, 1998.

[15] Ehsan Ullah Munir, Jianzhong Li, and Shengfei Shi. Qos sufferage heuristic for independent task scheduling in grid. *Information Technology Journal*, 6(8):1166–1170, 2007.

[16] M. Maheswaran, S. Ali, H.J. Siegal, D. Hensgen, and R.F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pages 30–44, 1999.

[17] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

[18] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, pages 349–363, 2000.

[19] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th*, pages 185–199, 2000.

[20] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001.

[21] Manzur Murshed, Rajkumar Buyya, and David Abramson. Gridsim: A toolkit for the modeling and simulation of global grids. Technical report, 2001.

[22] F. Azzedin and M. Maheswaran. Towards trust-aware resource management in grid computing systems. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, pages 452–452, 2002.

[23] F. Azzedin and M. Maheswaran. Evolving and managing trust in grid computing systems. In *Electrical and Computer Engineering, 2002. IEEE CCECE 2002. Canadian Conference on*, volume 3, pages 1424–1429 vol.3, 2002.

[24] B. Nazir and T. Khan. Fault tolerant job scheduling in computational grid. In *Emerging Technologies, 2006. ICET '06. International Conference on*, pages 708–713, 2006.

[25] Fatos Xhafa, Leonard Barolli, and Arjan Durresi. Batch mode scheduling in grid systems. *Int. J. Web Grid Serv.*, 3(1):19–37, March 2007.

[26] Zhan Gao, Siwei Luo, and Ding Ding. A scheduling mechanism considering simultaneous running of grid tasks and local tasks in the computational grid. In *Multimedia and Ubiquitous Engineering, 2007. MUE '07. International Conference on*, pages 1100–1105, 2007.

[27] Y. Demchenko, C. de Laat, O. Koeroo, and D. Groep. Re-thinking grid security architecture. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 79–86, 2008.

[28] Xiu mei Wen, Wei Zhao, and Fan xing Meng. Research of grid scheduling algorithm based on p2p grid model. In *Electronic Commerce and Business Intelligence, 2009. ECBI 2009. International Conference on*, pages 41–44, 2009.

[29] J. Bagherzadeh and M. MadadyarAdeh. An improved ant algorithm for grid scheduling problem. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 323–328, 2009.

[30] I. Rodero, F. Guim, and J. Corbalan. Evaluation of coordinated grid scheduling strategies. In *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on*, pages 1–10, 2009.

[31] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future Gener. Comput. Syst.*, 26(4):608–621, April 2010.

[32] 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. 4(10):1102–1109.