

# Design and Development of Scheduling Algorithms for Grid Computing Systems

**THESIS**

Submitted in partial fulfillment  
of the requirements for the degree of  
**DOCTOR OF PHILOSOPHY**

by

**Sunita Bansal**

Under the Supervision of  
**Prof. Chittaranjan Hota**



**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE  
PILANI (RAJASTHAN) INDIA**

January 2015

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE  
PILANI, RAJASTHAN**

**CERTIFICATE**

This is to certify that the thesis entitled "**Design and Development of Scheduling Algorithms for Grid Computing Systems**", submitted by **Sunita Bansal** ID No. **2006PHXF422P** for the award of Ph.D. degree of the Institute, embodies original work done by her under my supervision.

Signature of the supervisor :



Name : **DR. CHITTARANJAN HOTA**

Designation : Professor & Associate Dean,  
Department of Computer Science  
and Information Systems,  
BITS Pilani, Hyderabad Campus  
Hyderabad

Date :

23/02/2015

## Acknowledgements

I would like to express my gratitude to several personalities for their constant guidance, help, support, and well wishes.

I acknowledge with great pleasure my deep sense of gratitude to my supervisor Prof. Chittaranjan Hota for his constant encouragement, valuable guidance, and inspiring suggestions throughout the course of this work. At several times, research went into a despair state. At that crucial moment, talking to him for a few minutes inspired me and kept the work rolling. He gave the direction, and clues that made me go deeper into the work. He never let me stagnate but kept me on the move.

I would like to thank my DAC members, Prof. Sundar Shan Balasubramaniam and Prof. Santonu Sarkar for reviewing my proposal and for their valuable inputs. I would like to thank Prof. R. K. Mittal, Prof. G. Sunder, Prof. R. N. Saha, Prof. Hemant Ramanlal Jadhav, and Prof. S. Guru Narayanan for their constant encouragement and help. I would like to thank Prof. J. P. Mishra and Prof. Rahul Banerjee for encouraging me to complete my thesis. I would like to thank my students Divya, Gautam, Gino, Gowtham K., Abhishek, and Bhavik for their sincere project work.

Thanks are due to Prof. B. N. Jain, Vice-chancellor and Prof. G. Raghurama, Director for the constant support and concern. I would like to gratefully acknowledge Prof. Sanjay Kumar Verma, Dean, ARD and many other administrators for providing us suitable working atmosphere. I thank all my colleagues in our department and those known to me in other departments for their well wishes.

Last but not the least and most deeply, the author would like to thank her parents, her husband Mahesh, her son Parth, and other family members for their love and moral support during the entire period of this research work without which this work would not have been possible.

(Sunita Bansal)

## Abstract

Local resources available at a node are often insufficient to solve large computing problems. At the same time, underutilized resources remain unused because of ignorance of their capabilities, or incompatible administrative restrictions. To preserve the investment in equipment, and allow solving large computational problems, mechanisms are needed to join these independent systems into cooperating groups across the boundaries of administrative domains and physical proximity.

This cooperation is named as distributed computing that has many flavors like, Cloud computing, Grid computing, and Cluster computing. These distributed computing fields are concerned about aggregation of distributed computing power for solving large-scale problems in science, engineering, and commerce. However, application composition, resource management, and scheduling in these environments are complex undertakings. This is due to the geographic distribution of resources that are often owned by different organizations having different usage policies.

Due to the aggregation of heterogeneous resources, resource management is essential for Grid computing. This makes resource management in Grid systems distinct from traditional computation platform. Therefore, most task scheduling algorithms developed for traditional platforms are not applicable to Grid systems. Resource management includes searching, selecting, scheduling, and monitoring. This thesis focuses on scheduling aspect of Grid computing resource management while job submission, execution, and monitoring are delegated to user and provider middleware.

Efficiency of scheduling algorithms affects the user and service provider. Effectiveness of a scheduling algorithm is measured using response time, makespan, cost, deadline, budget, and communication overhead. A Grid scheduling algorithm is employed at two levels - local scheduling and global scheduling. Local scheduling algorithms manage the nodes within site and improve the system performance, while global scheduling algorithms select the site and improve the makespan and cost. They are of much relevance in these days because user has to pay-per-use.

In the thesis, various centralized scheduling algorithms have been developed and tested to improve makespan and cost; Decentralized scheduling algorithms are developed and tested to improve response time.

Contributions to the centralized scheduling algorithms are as follows:

Three centralized scheduling algorithms have been designed. First, a dependent task scheduling algorithm has been designed that works on economic Grid and tasks are scheduled using Double Hybrid Multi-objective Non-dominating Sorting Genetic Algorithm. The proposed algorithm minimizes three conflict objectives namely makespan, communication cost, and computation cost. This approach has 20% minimum objective value than other approaches. Second, Independent Parallel task scheduling algorithm has been designed that works on economic Grid. A Parallel task scheduling algorithm minimizes makespan, cost, and processor fragmentation simultaneously. It reduces overall average failure by 31%. Third, an Enhanced Refinery heuristic is designed for Independent task that works on computation Grid and reduces the makespan by 9% in case of an inconsistent matrix.

Contributions to the decentralized scheduling algorithms are as follows:

Two decentralized scheduling algorithms have been designed. First, an Efficient Dynamic Round Robin scheduling algorithm that models a scheduling algorithm as a state transition diagram and duplication candidate task is chosen intuitively to avoid impractical duplication. Overall response time is improved by 13% and 20% when job inter-arrival rate of tasks are large and small respectively. Enhanced Sender-initiated scheduling algorithm works on Grid system where nodes are heterogeneous in nature. It uses polling information to determine threshold. As a result, proposed approach decrease the 12% turnaround time and 23% network overhead.

# Contents

<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xii</b>
<b>List of Abbreviations/Symbols</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Characteristics of Grid Computing . . . . .	2
1.1.1 Heterogeneity . . . . .	2
1.1.2 Sharing of Resources . . . . .	2
1.1.3 Multiplicity of Administrative Domains . . . . .	2
1.1.4 Virtualization . . . . .	3
1.2 Categories of Grid . . . . .	3
1.2.1 Based on Scale . . . . .	3
1.2.1.1 Cluster Grid . . . . .	3
1.2.1.2 Campus Grid . . . . .	3
1.2.1.3 Global Grid . . . . .	4
1.2.2 Based on Function . . . . .	4
1.2.2.1 Computational Grid . . . . .	4
1.2.2.2 Data Grid . . . . .	4
1.2.2.3 Service Grid . . . . .	5
1.2.2.4 Utility Grid . . . . .	5
1.3 Distributed Environment . . . . .	5
1.3.1 Parallel Computing . . . . .	6
1.3.2 Distributed Computing . . . . .	7
1.3.3 Cluster Computing . . . . .	8
1.3.4 Grid Computing . . . . .	9
1.3.4.1 Information and Data Management . . . . .	11
1.3.4.2 Allocation of Resources . . . . .	11
1.3.4.3 Computational Economy . . . . .	11
1.3.5 Peer-to-Peer Computing . . . . .	12
1.3.6 Cloud Computing . . . . .	13

1.3.6.1	Software as a Service (SaaS)	13
1.3.6.2	Platform as a Service (PaaS)	14
1.3.6.3	Infrastructure as a Service (IaaS)	14
1.4	Comparison of Distributed and Parallel Systems	15
1.5	Resource Management in Cluster Computing	16
1.5.1	Condor	17
1.5.2	Portable Batch System	18
1.5.3	Load Leveler	19
1.5.4	Load Sharing Facility	19
1.5.5	Maui	20
1.6	Resource Management in Grid Computing	20
1.6.1	Gridbus	21
1.6.2	NetSolve	22
1.6.3	Legion	22
1.6.4	Condor-G	22
1.6.5	Nimrod-G	23
1.6.6	Askalon	24
1.6.7	Pegasus	25
1.7	Simulation Tools	26
1.7.1	Bricks	27
1.7.2	SimGrid	27
1.7.3	GridSim	28
1.7.4	Grid Scheduling Simulator (GSSIM)	28
1.8	Description of Problem	28
1.9	Thesis Contributions	32
1.10	Thesis Organization	33
<b>2</b>	<b>State of the Art</b>	<b>34</b>
2.1	Scheduling Algorithms in Distributed Computing	34
2.1.1	Sender-initiated Scheduling Algorithm	35
2.1.2	Receiver-initiated Scheduling Algorithm	35
2.1.3	Symmetrically-initiated Scheduling Algorithm	37
2.1.4	Stable Symmetrically-initiated Scheduling Algorithm	37
2.1.5	Stable Sender-initiated Adaptive Scheduling Algorithm	39
2.2	Scheduling Algorithms in Parallel/Cluster Computing	40
2.2.1	First-Come First-Served (FCFS) Scheduling Algorithm	41
2.2.2	Conservative Backfilling (CBF) Scheduling Algorithm	41
2.2.3	Aggressive Backfilling (ABF) Scheduling Algorithm	42
2.2.4	Easy Backfilling (EASY) Scheduling Algorithm	43
2.3	Scheduling Algorithms in Grid Computing	44
2.3.1	Objective Functions	45

2.3.1.1	Makespan . . . . .	45
2.3.1.2	Flow Time . . . . .	46
2.3.1.3	Average Resource Utilization Rate . . . . .	46
2.3.1.4	Load Balancing Level . . . . .	47
2.3.2	Categories of Task Scheduling Algorithms . . . . .	47
2.3.2.1	Dependent vs. Independent . . . . .	47
2.3.2.2	Online vs. Offline . . . . .	48
2.3.2.3	Meta-heuristics vs. Heuristics . . . . .	48
2.3.2.4	Resource Oriented vs. Application Oriented . . . . .	48
2.3.2.5	Single Objective vs. Multi-objective . . . . .	49
2.3.2.6	Best Effort vs. QoS Constraint . . . . .	49
2.3.3	Online Independent Task Scheduling Algorithms . . . . .	49
2.3.3.1	Random Scheduling Algorithm . . . . .	49
2.3.3.2	Round-Robin (RR) Scheduling Algorithm . . . . .	50
2.3.3.3	Optimistic Load Balancing (OLB) Scheduling Algorithm . . . . .	50
2.3.3.4	Minimum Execution Time (MET) Scheduling Algorithm . . . . .	50
2.3.3.5	Minimum Completion Time (MCT) Scheduling Algorithm . . . . .	50
2.3.4	Offline Independent Task Scheduling Algorithms . . . . .	51
2.3.4.1	Min-min Heuristic . . . . .	51
2.3.4.2	Max-min Heuristic . . . . .	51
2.3.4.3	Suffrage Heuristic . . . . .	51
2.3.4.4	QoS Guided Min-min Heuristic . . . . .	52
2.3.4.5	High Standard Deviation First Heuristic . . . . .	52
2.3.4.6	Segmented Min-min Heuristic . . . . .	52
2.3.4.7	Resources Aware Scheduling Algorithm (RASA) . . . . .	53
2.3.4.8	Preemptive Version of Min-min Heuristic . . . . .	53
2.3.4.9	Modified Minimum Completion Time (MMCT) Scheduling Algorithm . . . . .	53
2.3.4.10	Min-mean Heuristic . . . . .	53
2.3.4.11	Refinery Heuristic . . . . .	54
2.3.5	Dependent Task Scheduling Algorithms . . . . .	54
2.3.5.1	HEFT Scheduling Algorithm . . . . .	54
2.3.5.2	Cluster and Duplication Based Scheduling Algorithms . . . . .	55
2.3.5.3	Dynamic Workflow Scheduling Algorithms . . . . .	56
2.3.6	Meta-heuristic . . . . .	56
2.3.6.1	Simulated Annealing (SA) . . . . .	56
2.3.6.2	Genetic Algorithms (GA) . . . . .	57
2.3.6.3	Combined Heuristics . . . . .	58
2.3.7	Approaches of Meta-heuristic . . . . .	58
2.3.8	Multi-objective Meta-heuristic . . . . .	59
2.3.8.1	Principle of Multi-objective Optimization (MOO) . . . . .	59



2.3.8.2	Vector Evaluated Genetic Algorithm (VEGA) . . . . .	61
2.3.8.3	Strength Pareto Evolutionary Algorithm (SPEA) . . . . .	61
2.3.8.4	Strength Pareto Evolutionary Algorithm 2 (SPEA2) . . . . .	61
2.3.8.5	Pareto Archived Evolution Strategy (PAES) Algorithm . . . . .	62
2.3.8.6	Non-dominated Sorting Genetic Algorithm (NSGA) . . . . .	62
2.3.8.7	Non-dominated Sorting Genetic Algorithm-II (NSGA-II) . . . . .	62
2.3.9	Approach of Multi-objective Meta-heuristics . . . . .	63
2.3.10	Utility/Quality of Service Based Scheduling Algorithms . . . . .	64
2.3.11	Approach of QoS Based Scheduling Algorithms . . . . .	64
2.3.12	Comparison of Scheduling Algorithms . . . . .	67
2.3.13	Shortcoming of Existing Scheduling Algorithms . . . . .	67
<b>3</b>	<b>Dependent Task Scheduling</b>	<b>70</b>
3.1	Motivation . . . . .	70
3.1.1	Steps of NSGA-II . . . . .	71
3.1.2	Crowded Comparison . . . . .	71
3.2	Problem Definition . . . . .	72
3.2.1	Objective functions . . . . .	75
3.3	Double Hybrid NSGA-II (DHNSGA-II) . . . . .	75
3.3.1	Implementation of DHNSGA-II . . . . .	77
3.3.1.1	Chromosome Representation . . . . .	77
3.3.1.2	Selection/Replacement . . . . .	79
3.3.1.3	Crossover . . . . .	79
3.3.1.4	Mutation . . . . .	79
3.3.1.5	Pre-selection . . . . .	80
3.3.1.6	Local Search/Memetic Operator . . . . .	80
3.3.1.7	Evaluation . . . . .	81
3.3.2	Time Complexity . . . . .	81
3.4	Simulation and Evaluation . . . . .	81
3.4.1	Performance Index . . . . .	83
3.4.2	Results of DHNSGA-II . . . . .	83
3.4.3	Ranking of Non-dominated Solutions . . . . .	85
3.4.4	Results of Ranking Algorithm . . . . .	86
3.5	Discussion . . . . .	87
<b>4</b>	<b>Independent Task Scheduling</b>	<b>89</b>
4.1	Enhanced Refinery Heuristic . . . . .	89
4.1.1	Motivation . . . . .	89
4.1.2	Proposed Enhanced Refinery Heuristic . . . . .	92
4.1.3	Illustrative Example of Enhanced Refinery Heuristic . . . . .	94
4.1.4	Experimental Setup . . . . .	97

4.1.4.1	Constructing ETC Matrix . . . . .	97
4.1.4.2	Achieving Heterogeneity . . . . .	98
4.1.5	Experimental Results . . . . .	99
4.2	Parallel Task Scheduling Algorithm . . . . .	103
4.2.1	Motivation . . . . .	103
4.2.2	System Model . . . . .	104
4.2.3	Problem Statement . . . . .	105
4.2.4	EMCT Scheduling Algorithm . . . . .	107
4.2.5	TOPSIS Algorithm . . . . .	108
4.2.5.1	Construction of Decision Matrix D . . . . .	108
4.2.6	Simulation and Evaluation . . . . .	109
4.3	Discussion . . . . .	115
<b>5</b>	<b>Decentralized Task Scheduling</b>	<b>116</b>
5.1	Motivation . . . . .	116
5.2	EDRR Scheduling Algorithm . . . . .	117
5.2.1	Assumptions . . . . .	117
5.2.2	Proposed Solution . . . . .	118
5.2.2.1	Waiting State . . . . .	119
5.2.2.2	Execution State . . . . .	119
5.2.2.3	Replica State . . . . .	120
5.2.2.4	Removal of Replica . . . . .	121
5.2.3	Experimental Setup . . . . .	122
5.2.4	Experimental Results . . . . .	122
5.3	ESender Scheduling Algorithm . . . . .	125
5.3.1	Information Policy . . . . .	126
5.3.2	Transfer Policy . . . . .	126
5.3.3	Selection Policy . . . . .	126
5.3.4	Location Policy . . . . .	126
5.3.5	Simulation and Evaluation . . . . .	127
5.4	Discussion . . . . .	130
<b>6</b>	<b>Conclusions</b>	<b>131</b>
6.1	Conclusions . . . . .	131
6.2	Summary of Contributions . . . . .	132
6.3	Future Research . . . . .	133
<b>A</b>	<b>Simulator Input-Output</b>	<b>134</b>
A.1	DHNSGA-II Input-Output . . . . .	134
A.1.1	Input . . . . .	134
A.1.2	Output . . . . .	139
A.2	Enhanced Refinery (ER) Heuristic Input-Output . . . . .	144

A.2.1	Input . . . . .	144
A.2.2	Output . . . . .	145
	<b>References</b>	<b>150</b>
	<b>Publications</b>	<b>164</b>
	<b>Biographies</b>	<b>167</b>

# List of Figures

1.1	Parallel Computing . . . . .	6
1.2	Distributed Computing . . . . .	8
1.3	Cluster Computing . . . . .	9
1.4	Grid Computing . . . . .	10
1.5	A Layered Grid Architecture . . . . .	12
1.6	Peer-to-Peer Computing . . . . .	13
1.7	Cloud Computing . . . . .	14
1.8	Condor Matchmaking Process . . . . .	17
1.9	Portable Batch System . . . . .	19
1.10	Grid Resource Management Topology . . . . .	21
1.11	Legion Resource Management . . . . .	23
1.12	Askalon Scheduler Architecture . . . . .	25
1.13	Pegasus Scheduler Architecture . . . . .	26
1.14	Grid Scheduling Architecture . . . . .	29
1.15	Workflow Partitioning Techniques . . . . .	31
2.1	Working of Sender-initiated Scheduling Algorithm . . . . .	36
2.2	Working of Receiver-initiated Scheduling Algorithm . . . . .	36
2.3	First-Come First-Served Example . . . . .	41
2.4	Conservative Backfilling Example . . . . .	42
2.5	Aggressive Backfilling Example . . . . .	43
2.6	Easy Backfilling Example . . . . .	44
2.7	Classification of Objective Functions . . . . .	45
2.8	Unconstrained Non-dominated . . . . .	60
3.1	Workflow of 8 Tasks . . . . .	73
3.2	Pictorial Diagram DHNSGA-II . . . . .	77
3.3	Two Point Crossover . . . . .	80
3.4	Comparison of Seeded NSGA-II Solutions with Reference Solutions . . . . .	84
3.5	Comparison of Memetic NSGA-II Solutions with Reference Solutions . . . . .	84
3.6	Comparison of DHNSGA-II Solutions with Reference Solutions . . . . .	85
4.1	ETC Matrix . . . . .	95

4.2	Min-min Heuristic . . . . .	95
4.3	Iteration-1 of ER Heuristic (Swap Procedure) . . . . .	96
4.4	Iteration-1 of ER Heuristic (Move Procedure) . . . . .	96
4.5	Iteration-2 of ER Heuristic (Swap Procedure) . . . . .	96
4.6	Iteration-2 of ER Heuristic (Move Procedure) . . . . .	96
4.7	Iteration-3 of ER Heuristic (Move Procedure) . . . . .	96
4.8	Average Makespan of lo-lo Heterogeneity . . . . .	101
4.9	Average Makespan of hi-hi Heterogeneity . . . . .	101
4.10	Average Makespan of lo-hi Heterogeneity . . . . .	102
4.11	Average Makespan of hi-lo Heterogeneity . . . . .	102
4.12	Improvement of ER Heuristic Over Refinery Heuristic . . . . .	103
4.13	Grid Topology . . . . .	110
4.14	Average Cost with Number of Applications . . . . .	111
4.15	Average Makespan with Number of Applications . . . . .	112
4.16	Average Failure with Number of Applications . . . . .	112
4.17	Average Cost with Varying Trade-off Factor . . . . .	113
4.18	Average Makespan with Varying Trade-off Factor . . . . .	114
4.19	Average Failure with Varying Trade-off Factor . . . . .	114
5.1	EDRR Scheduling Algorithm Grid Model . . . . .	118
5.2	Comparison of DRR and EDRR Scheduling Algorithm for Case I . . . . .	123
5.3	Comparison of DRR and EDRR Scheduling Algorithm for Case II . . . . .	123
5.4	Comparison of DRR and EDRR Scheduling Algorithm for Case III . . . . .	124
5.5	Comparison of DRR and EDRR Scheduling Algorithm for Case IV . . . . .	124
5.6	Flowchart of ESender Scheduling Algorithm . . . . .	128
5.7	Comparison of Number of Messages Transfer of SI and ESender Scheduling Algorithm . . . . .	129
5.8	Comparison of Turnaround Time with SI of ESender Scheduling Algorithm	129
A.1	Task Graph Generator . . . . .	134
A.2	Gauss Elimination Graph of Matrix Size of 8 . . . . .	136

# List of Tables

1.1	Comparison of Distributed and Parallel Computing . . . . .	7
1.2	Different Types of Distributed Computing Systems . . . . .	16
1.3	Simulator Classification . . . . .	27
2.1	Multi-objective Optimization Algorithms . . . . .	63
2.2	Evolution of Scheduling Algorithms . . . . .	68
2.3	Overview of Various Scheduling Algorithms . . . . .	68
3.1	Workflow Parameters . . . . .	82
3.2	DHNSGA-II Parameters . . . . .	82
3.3	HV of Different Matrices . . . . .	85
3.4	Ratio of Makespan and Total Cost of Matrix Size 32 . . . . .	87
3.5	Ratio of Makespan and Total Cost of Matrix Size 64 . . . . .	87
4.1	Excerpt from Inconsistent High Heterogeneity of Tasks and Machine . . . . .	99
4.2	Excerpt from Semi High Heterogeneity of Tasks and Machine . . . . .	99
4.3	Notations . . . . .	106
4.4	Decision Matrix D . . . . .	109
4.5	Grid Resources . . . . .	109

## List of Abbreviations/Symbols

<b>Term</b>	<b>Definition</b>
QoS	Quality of Service
SLA	Service Level Agreements
FIFO	First-In-First-Out
CPU	Central Processing Unit
HTC	High Throughput Clusters
HPC	High Performance Clusters
LHC	Large Hadron Collider
P2P	Peer-to-Peer
NIST	National Institute of Science and Technology
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
ClassAds	Classified Advertisement
PBS	Portable Batch System
MOM	Machine Oriented Miniserver
SJF	Shortest Job First
LL	Load Leveler
LSF	Load Sharing Facility
GSI	Grid Security Infrastructure
GRAM	Grid Resource Allocation and Management
GIS	Grid Information Service
SAN	Storage Area Networks
NFS	Network File Systems
DSS	Dedicated Storage Servers
VD	Virtual Database
API	Application Programming Interface
TFE	Task Farming Engine
DAG	Directed Acyclic Graph
AGWL	Abstract Grid Workflow Language
GridARM	Grid Askalon Resource Manager
MDS	Monitoring and Discovery Service
RLS	Replica Location Service
DAGMan	Directed Acyclic Graph Manager
PWME	Pegasus Workflow Mapping Engine
VDL	Virtual Data Language

TC	Transformation Catalog
NorduGrid	Nordic Testbed for Wide Area Computing and Data Handling
ARC	Advanced Resource Connector
EGEE	Enabling Grids for E-science in Europe
AR	Advance Reservation
NWS	Network Weather Service
HEP	High-Energy Physics
GSSIM	Grid Scheduling Simulator
SWF	Standard Workload Format
GWF	Grid Workload Format
NSGA	Non-dominating Sorting Genetic Algorithm
FCFS	First-Come First-Served
CBF	Conservative Backfilling
ABF	Aggressive Backfilling
EASY	Easy Backfilling
HEFT	Heterogeneous Earliest-Finish-Time
MCT	Minimum Completion Time
MET	Minimum Execution Time
SPEA2	Strength Pareto Evolutionary Algorithm
PEAS	Pareto Archived Evolution Strategy
PSO	Particle Swarm Optimization
SNSGA	Steady-State NSGA-II
OLB	Optimistic Load Balancing
ETC	Expected Time to Compute
RT	Ready Time
HSDF	High Standard Deviation First
RASA	Resources Aware Scheduling Algorithm
MMCT	Modified Minimum Completion Time
DAG	Directed Acyclic Graph
FCP	Fast Critical Path
IPC	Inter Process Communication
TDS	Task Duplication based Scheduling
EST	Earliest Start Time
ECT	Earliest Completion Time
LAST	Latest Allowable Start Time
LACT	Latest Allowable Completion Time
LT	Level of Task
FP	Favorite Predecessor
GA	Genetic Algorithms



SA	Simulated Annealing
ACO	Ant Colony Optimization
GSA	Genetic Simulated Annealing
GrADS	Grid Application Development Software
MOO	Multi-objective Optimization
VEGA	Vector Evaluated Genetic Algorithm
EMOO	Evolutionary Multi-objective Optimization
NSGA-II	Non-dominated Sorting Genetic Algorithm-II
RNSGA-II	Referenced Point NSGA-II
DBC	Deadline and Budget Constrained
GRM	Grid Resource Manager
DRM	Domain Resource Manager
CN	Computing Node
MinCTT	Min-min Cost Time Trade-off
MaxCTT	Min-max Cost Time Trade-off
SuffrageCTT	Suffrage Cost Time Trade-off
DHNSGA-II	Double Hybrid NSGA-II
TOPSIS	Technique for Order Performance by Similarity to Ideal Solution
MCDM	Multiple Criteria Decision Making
AHP	Analytic Hierarchy Process
IQR	Inter Quartile Range
ER	Enhanced Refinery
GIS	Grid Information Server
MIPS	Million Instructions Per Second
SIMD	Single Instruction Multiple Data
EMCT	Economic Minimum Completion Time
PE	Processing Elements
EDRR	Efficient Dynamic Round Robin
DRR	Dynamic Round Robin
ESender	Enhanced Sender-initiated
SI	Sender-initiated
RR	Round-robin
AWS	Amazon Web Services
ETC	Expected Time to Compute
hi-hi	High Task Heterogeneity and High Machine Heterogeneity
hi-lo	High Task Heterogeneity and Low Machine Heterogeneity
lo-hi	Low Task Heterogeneity and High Machine Heterogeneity
lo-lo	Low Task Heterogeneity and Low Machine Heterogeneity
<i>compuCost</i>	Computation Cost

---

<i>commuCost</i>	Communication Cost
<i>popSize</i>	Population Size
<i>gen</i>	Generation
<i>Ss</i>	Solution String
<i>Ms</i>	Matching String
<i>c</i>	Current Solution
<i>M<sub>m</sub></i>	Makespan Machine
$\Psi$	Execution Time
$\alpha$	Response Time
$\tau$	Transfer Time
<i>T</i>	Threshold
<i>G\$</i>	Grid Dollar

# Chapter 1

## Introduction

Several factors like resource sharing, scalability, etc. have taken communication to the era of Grid computing. This permits desktop computers to take part in a global network activity when they are idle, and it enables large software systems to utilize extra hardware resources. Like the human brain, modern computers normally utilize only a small portion of their potential and are typically inactive while awaiting inbound tasks. When all the resources of inactive computer systems are gathered as an all-in-one computer system, a highly effective system arises.

With the assistance of the Internet, Grid computing has supplied the ability to utilize hardware resources that belong to various other systems. Grid computing may have different definitions for various individuals, however, as a simple interpretation, Grid computing is a system that permits us to link network resources and application programs and make a large effective system that has the capability to do extremely complex jobs that a solitary personal computer could not complete. That is, from the perspective of the users of Grid systems, these operations can only be performed through these systems. As large infrastructure for parallel and distributed computing systems, Grid systems allow the virtualization of a vast array of resources, in spite of their considerable heterogeneity.

Grid computing has numerous benefits for developers and administrators. For instance, Grid computing systems can operate programs that need a large amount of memory and can make information simpler for accessing. Grid computing could help large organizations and firms that have actually made a substantial investment to benefit from their systems. Therefore, Grid computing has attracted the attention of industrial man-

agers and investors in companies that have become involved in Grid computing, such as IBM, HP, Intel, and Sun. By focusing on resource sharing and coordination, managing capabilities, and attaining high efficiency, Grid computing has become an important component of the computer industry. However, it is still in the developmental stage, and several issues and challenges remain to be resolved. Grid differs from traditional parallel and distributed system because these systems are usually homogeneous and dedicated. Scheduling algorithms that are designed for these systems can not work well in Grid system due to following reasons:

- Resources reside within a single administration domain.
- Scheduler has knowledge of other resources status.
- Resources are static.
- Communication cost is negligible.

In the next section we discuss the characteristics of Grid computing.

## **1.1 Characteristics of Grid Computing**

### **1.1.1 Heterogeneity**

Grid is a collection of parallel and distributed system that are connected on wide area network and belongs to multiple domains. Thus, it requires to address storage, computation, and communication heterogeneity.

### **1.1.2 Sharing of Resources**

Resources in a Grid belong to many different organizations that allows harness of the ideal resources. These resources are shared to increase the efficiency and decrease the cost.

### **1.1.3 Multiplicity of Administrative Domains**

Each organization can establish management policies and different security techniques to control the usage of resources deployed in a Grid in a secure manner. Resources must be

accessible and usable by all customers of the Grid. Thus, it requires a method to provide secure and reliable access of Grid.

#### **1.1.4 Virtualization**

Grid system creates virtual resources based on the problem and availability of resources. These entities are of limited life, dynamically created and are used to solve the problem.

## **1.2 Categories of Grid**

Grid computing is classified based on the structure of the organization that is served and based on principle for which resources are used in the Grid. For example, Campus Grid served within campus and Computational Grid efficiently executes the submitted jobs. Grid categories are described in section 1.2.1 and 1.2.2.

### **1.2.1 Based on Scale**

#### **1.2.1.1 Cluster Grid**

Cluster Grid is the most popular and simplest form of a Grid. A cluster Grid consists of one or more systems, working together, to provide a single point of access to users. Cluster Grid meets the need of most of the organizations. Typically it is used by a team of users such as people working in a single project or in a department. A cluster Grid supports high throughput.

#### **1.2.1.2 Campus Grid**

Campus Grid enables multiple projects or departments to share computing resources in a cooperative way. It is also referred as the cooperative Grid. Campus Grid may consist of dispersed workstations and servers, as well as centralized resources located in multiple administrative domains, in departments, or across the enterprise.

### 1.2.1.3 Global Grid

When an application needs to exceed the capacity of a campus Grid, organizations can tap partner resources through a global Grid. Designed to support and address the needs of multiple sites and organizations, global Grid provides the power of distributed resources to users anywhere in the world for computing and collaboration. Individuals or organizations sending excess work to a Grid provider or multiple companies working together and sharing data-crossing organizational boundaries with ease can use the global Grid.

## 1.2.2 Based on Function

### 1.2.2.1 Computational Grid

The computational Grid systems have higher aggregate computational capacity for single applications than the capacity of any constituent machine in the system. Depending on how this capacity is utilized, these systems can be further subdivided into distributed supercomputing and high throughput categories. A distributed supercomputing Grid executes the application in parallel on multiple machines to reduce the completion time of a job. Typically, applications that require distributed supercomputing are grand challenge problems such as weather modeling and nuclear simulations. The computation Grid increases the completion rate of parameter sweep type applications (Buyya *et al.* 2002).

### 1.2.2.2 Data Grid

Data Grid provides an infrastructure for synthesizing new information from data repositories such as digital libraries or data warehouses that are distributed in a wide area network. Computational Grids also need to provide data services, but the major difference between a data Grid and a computational Grid is the specialized infrastructure provided to applications for storage management and data access. In a computational Grid, applications implement their own storage management schemes rather than use Grid provided services. Typical applications, which include special purpose data mining activities that correlate information from multiple data sources. The data Grid initiatives, European Data Grid Project (Hoscheck *et al.* 2000) and Globus (Chervenak *et al.* 2000), are working on developing large-scale data organization, catalogue, management, and access

technologies.

### 1.2.2.3 Service Grid

The service Grid is a smart, end-to-end service integration platform which provides services to the user. This category is further subdivided as on-demand, collaborative, and multimedia Grid systems. A collaborative Grid connects users and applications into collaborative work-groups. These systems enable real time interaction between humans and applications via a virtual workspace. An on-demand Grid dynamically aggregates different resources to provide new services. A data visualization workbench that allows a scientist to dynamically increase the fidelity of a simulation by allocating more machines to a simulation would be an example of an on-demand Grid system. A multimedia Grid provides an infrastructure for real-time multimedia applications. This requires supporting Quality of Service (QoS) across multiple different machines, whereas a multimedia application on a single dedicated machine may be deployed without QoS (Nahrstedt *et al.* 1998).

### 1.2.2.4 Utility Grid

The utility Grid environment can be considered as a market where competition takes place between consumer and providers. Consumer wants to execute his/her task at least cost and in less time. Providers lease their resources in order to earn revenue. The creation of utility Grid requires the integration of scalable system architecture, resource management, scheduling, and market models. In order to make the consumers participate in the utility Grid, mechanisms for bidding and cost minimization are required. Utility Grid is different from community Grid. Community Grid provides free access, whereas users need to pay for service access in utility Grid. In utility Grid, users can make a reservation with a service provider in advance to ensure the service availability and users can also negotiate with service providers on Service Level Agreements (SLA) for getting the required QoS.

## 1.3 Distributed Environment

For any compute intensive job, it is desired that the submitted job gets executed in minimum time. The advent of multi-processor and multi-computer systems has ensured this goal effectively. The effort towards parallel/concurrent computing has resulted in Parallel, Distributed, Cluster, Grid, Peer-to-Peer, and Cloud computing. This section discusses these platforms in brief.

### 1.3.1 Parallel Computing

Parallel computing is referred to as a tightly coupled system. It has a collection of processors, memory and common system bus as shown in Fig. 1.1. It communicates through shared memory. It can be classified into asymmetric and symmetric systems. Symmetric multiprocessor involves all the processors to process jobs, whereas in asymmetric system, one processor acts as a master, others act as slaves. Master processor allocates jobs to slave processors. The problem with master/slave configuration is that master processor will become a bottleneck at the time of peak load.

Tightly coupled scheduling systems have one more dimension than a single processor system. Multiprocessor scheduler selects a process as well as a processor. Processor can process unrelated or related processes. Each unrelated process runs independently, whereas related process runs in a group. In tightly coupled systems, all processes reside in the same memory. Whenever a CPU finishes its current task, it picks a new process. Unrelated processes are scheduled according to their time sharing requirement. Major scheduling algorithms under this category are Affinity scheduling algorithm (Singhal & Shivaratri 1998) and Smart scheduling algorithm (Singhal & Shivaratri 1998). Related processes use Space sharing scheduling algorithm. Examples of Space sharing scheduling algorithms are First-In-First-Out (FIFO) (Muallem & Feitelson 2001), Backfilling (Muallem & Feitelson 2001), Conservative Backfilling (Muallem & Feitelson 2001), Aggressive (Muallem & Feitelson 2001), and Gang scheduling (Ousterhout 1982).



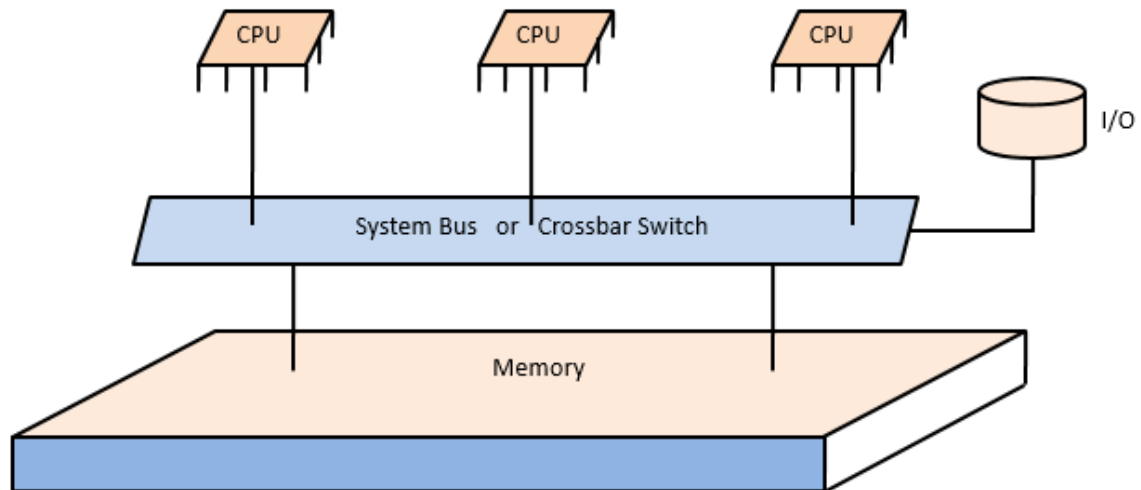


Figure 1.1: Parallel Computing - Source: (Tanenbaum & Bos 2015)

### 1.3.2 Distributed Computing

Distributed computing is also known as loosely coupled system. It is a collection of computers (CPU, memory, secondary storage, etc.) instead of only a single computer. They communicate through message passing over a network as shown in Fig. 1.2. In a distributed system each node has its own memory, set of processes, and a local scheduling algorithm.

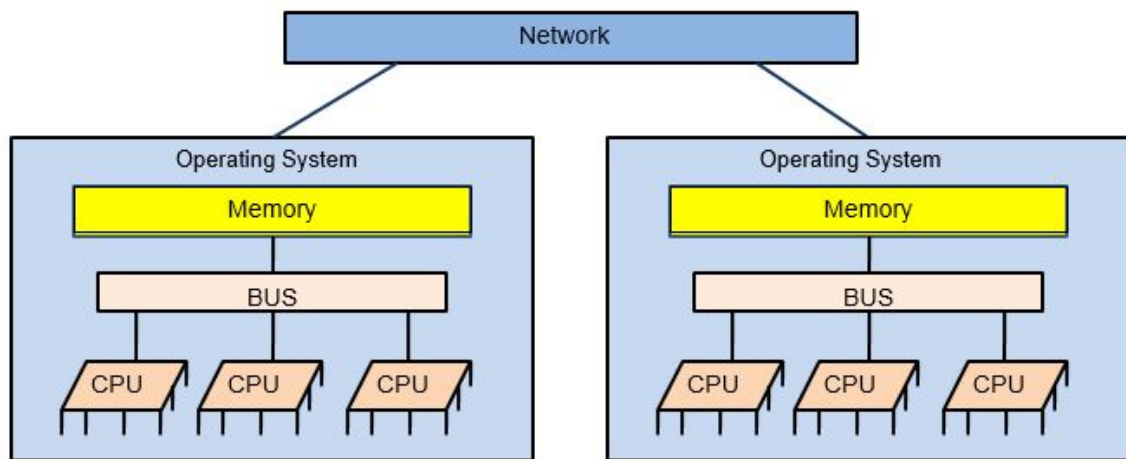
Distributed scheduling algorithms achieve better system performance by smoothing out any workload imbalance that may exist in a distributed system, such as minimizing communication delays, minimizing execution time, and maximizing resource utilization (Kureger & Livny 1987). Load scheduling is the process of deciding where to execute a process in a multi-computer system. This can be carried out by a single authority (Singhal & Niranjana 2006) or by many entities (Chou & Abraham 1982), (Krueger & Finkel 1984), (Shivaratri *et al.* 1992) spread in a distributed system. It also decides whether to equalize the load at all the computers or to share the load between highly loaded to lightly loaded nodes (Chou & Abraham 1982).

Table 1.1 depicts the comparison between distributed and parallel computing. Distributed computing refers to site autonomy where a node is free to behave differently than other nodes in the system, whereas a node cannot behave differently in parallel computing. Distributed computing has global scheduling and local scheduling. Global

**Table 1.1:** Comparison of Distributed and Parallel Computing

Distributed Computing	Parallel Computing
Global and Local scheduling	Local scheduling
Site autonomy	No site autonomy
Communication through message passing	Communication through shared memory

scheduling assigns a task to any processor within the system which refers to task mapping or task placement. Objectives of global scheduling are load balancing and minimization of communication delay. Local scheduling refers to Central Processing Unit (CPU) scheduling. Minimization of waiting time and turnaround time are the objectives of local scheduling.

**Figure 1.2: Distributed Computing** - Source: (Tanenbaum & Bos 2015)

Due to advancement in CPU computing power and communication bandwidth, traditional distributed computing emerged as Cluster, Grid, Cloud, and Peer-to-Peer computing. These modules are described in the next section.

### 1.3.3 Cluster Computing

Clusters are usually deployed to improve performance and availability over single computers. It creates an illusion of being a single machine. It can be a collection of multi-computer or multiprocessor systems as shown in Fig. 1.3. Clusters are divided into two major classes: High Throughput Clusters (HTC) and High Performance Clusters (HPC). HTC usually connect a large number of nodes using low-end interconnects. In contrast, HPC connect more powerful compute nodes using faster interconnects. Fast in-

terconnects are designed to provide lower latency and higher bandwidth than low-end interconnects. Examples of cluster computers are Beowulf (Becker & Sterling 1995), Sun cluster (Sun 2014), and Windows cluster (Win 2014).

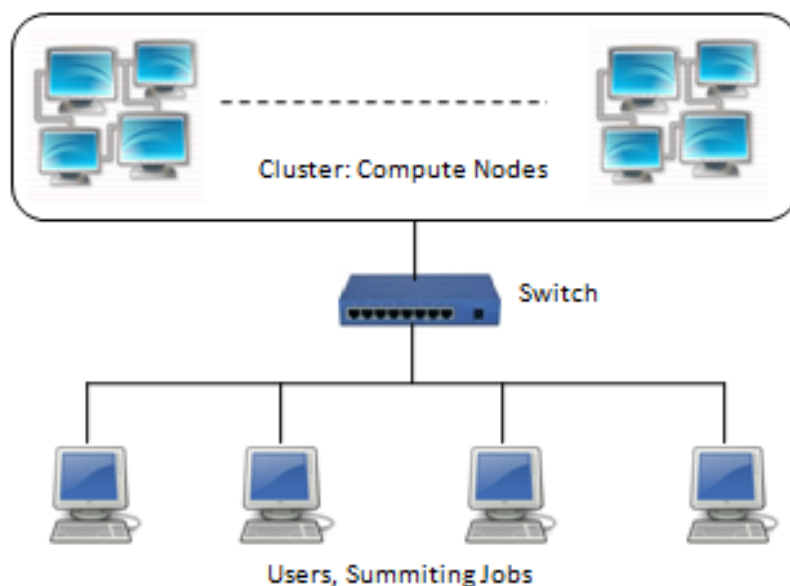


Figure 1.3: Cluster Computing - Source: (Clu 2015)

These two classes of clusters have different scheduling requirements. In HTC computing, the main goal is to maximize throughput, jobs completed per unit time by reducing load imbalance among compute nodes in the cluster. Load balancing is particularly important if the cluster has heterogeneous compute nodes. In HPC, an additional consideration arises: the need to minimize communication overhead by mapping applications appropriately to the available compute nodes. HTC are suitable for executing loosely coupled parallel or distributed applications, because such applications do not have high communication requirements among compute nodes during execution time. High-performance computing clusters are more suitable for tightly coupled parallel applications, which have substantial communication and synchronization requirements.

A resource management system manages the processing load by preventing jobs from competing with each other for limited compute resources. Typically, a resource management system comprises of a resource manager and a job scheduler. The scheduler communicates with the resource manager to obtain information about queues, loads on compute nodes, and resource availability to make scheduling decisions. Scheduling algorithms

can be broadly divided into two classes: time-sharing and space-sharing. Time-sharing algorithms divide time of a processor into several discrete intervals, or slots. These slots are then assigned to unique jobs. Hence, several jobs at any given time can share the same compute resources. Conversely, space-sharing algorithms give the requested resources to a single job until the job completes the execution. Most cluster schedulers operate in space-sharing mode. The most common space sharing scheduling algorithms are First-In-First-Out and Round-robin. There are various commercial resource managers like Maui (Jackson *et al.* 2001), Portable batch system (Yan & Chapman 2005), and Condor (Thain *et al.* 2005) available today in the market.

### 1.3.4 Grid Computing

Grid computing is inspired by the electrical power Grid. Looking at the ease of use, pervasiveness and reliability of the electrical power Grid, computer scientists too started exploring the design/development of an analogous infrastructure for wide-area parallel and distributed computing and data sharing. Simplest form of Grid as shown in Fig. 1.4 where resources are connected over wide area network to serve various users. The motivation for Grids was initially driven by large-scale resource (computational and data) intensive scientific applications that required more resources than a single computer (PC, workstation, supercomputer, or cluster) could provide to a single administrative domain. Grid computing strives to aggregate diverse, heterogeneous, and geographically distributed and multiple domain spanning resources to provide a platform for transparent, secure, coordinated, and high-performance resource-sharing and problem solving platform.

A Grid layered architecture along with the services provided by each layer is presented in Fig. 1.5 with connectivity layer, collective services layer, and resource layer together represented as core Grid middleware (Baker *et al.* 2002). Submission of jobs corresponding to various applications is represented as Application layer. Applications can be developed using Grid-enabled languages and utilities like HPC++ or MPI. The user level Grid Middleware includes application development environments, programming tools, and resource brokers for managing resources and scheduling application tasks for execution on global resources. The core Grid Middleware offers services such as remote process man-

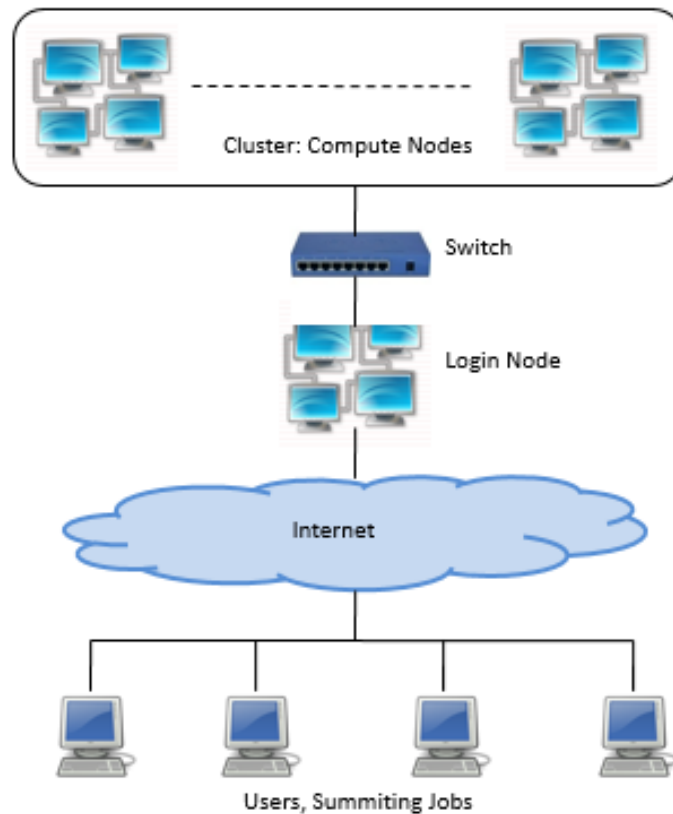


Figure 1.4: Grid Computing - Source: (Gri 2015)

agement, co-allocation of resources, storage access, information registration and discovery, security, and QoS ensuring. The Fabric layer corresponds to the computational resources held by various participants. The job of the Grid middleware is to act as an interface between the user and the Grid and to provide a homogeneous view of the heterogeneous Grid to the participants while providing the following services:

#### 1.3.4.1 Information and Data Management

The Grid Middleware (GM) should allow the various available resources to enroll themselves and communicate their services to the entire pool. Once these resources become part of the pool and an application is allotted on these resources the GM should ensure its secured execution while meeting the agreed Quality of Service (QoS) requirements. Further, the GM should provide secured services to create, manage, and access data sets involved in the computation.

#### 1.3.4.2 Allocation of Resources

When a user connects to a Grid, he simply specifies his application and its requirements. It is the job of the GM to provide access to resources, CPU time, memory, network bandwidth, and other components in order to extract the best performance from them.

#### 1.3.4.3 Computational Economy

There could be administrative domains in which the resources owner may provide rent based services. The GM should be able to meter the usage of such resources and accordingly realizing the payment. This enables the resources to be chosen according to their prices.

Resource scheduling in computational Grids has an important role in improving the efficiency. The Grid environment is very dynamic, with the number of resources, their availability, CPU loads, and the amount of unused memory constantly changing. In addition, different tasks have distinct characteristics that require different schedules. For instance, some tasks require high processing speeds and may require a great deal of coordination between their processes. Examples of Grid computing are SETI@home Project (Set 2014) and Large Hadron Collider (LHC) (Lhc 2014) Grid computing.

#### 1.3.5 Peer-to-Peer Computing

Peer-to-Peer (P2P) (Foster & Iamnitchi 2003) computing represents computing over an application layer, wherein all interactions among the processors are at a "peer" level, without any hierarchy among the processors. Thus, all processors are equal and play a symmetric role in the computation. P2P computing arose as a paradigm shift from client-server computing, where the roles among the processors are essentially asymmetrical. P2P networks are typically self-organizing, and may or may not have a regular structure of the network as shown in Fig. 1.6. No central directories (such as those used in Domain Name Servers) for name resolution and object lookup are allowed. Some of the key challenges in this paradigm include: object storage mechanisms, efficient object lookup and retrieval in a scalable manner; dynamic reconfiguration with nodes as well as objects joining and leaving the network randomly; replication strategies to expedite object search; tradeoffs

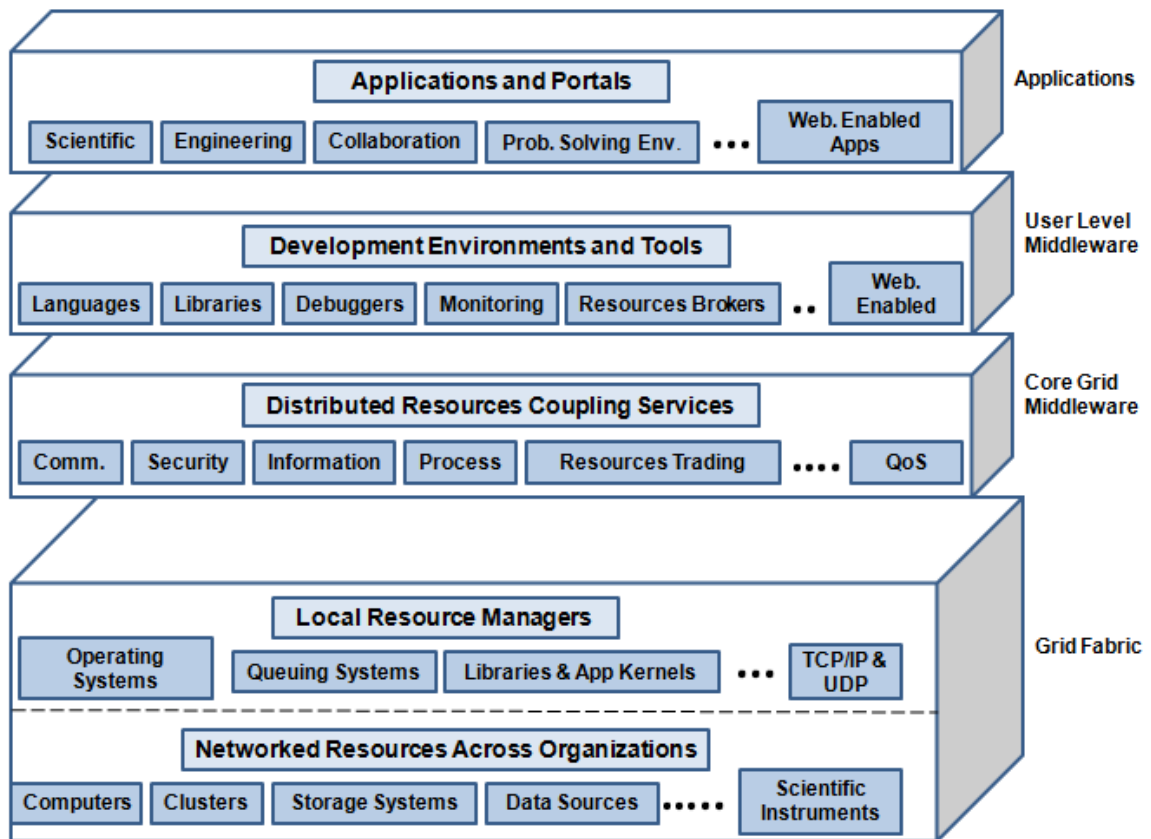


Figure 1.5: A Layered Grid Architecture - Source: (Buyya *et al.* 2002)

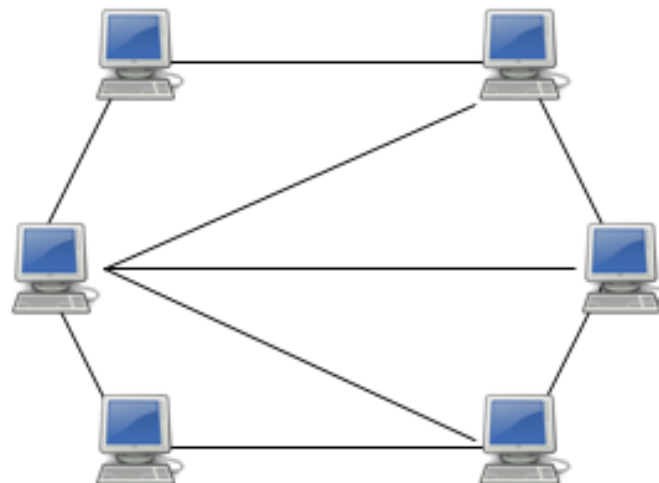


Figure 1.6: Peer-to-Peer Computing - Source: (Pee 2015)

between object size latency and table sizes; anonymity, privacy, and security. Examples of P2P computing are Bittorrent (Bit 2014a), Skype (Sky 2014), and Bitvault (Bit 2014b).

### 1.3.6 Cloud Computing

Cloud computing provides seamless and unlimited facilities. It is similar to Grid computing. It refers to the hardware and systems software in the data centers that provide computing resources as services. It provides three kinds of service (Clo 2014a) models defined by the National Institute of Science and Technology (NIST) as shown in Fig. 1.7 and describes in section 1.3.6.1 to 1.3.6.3.

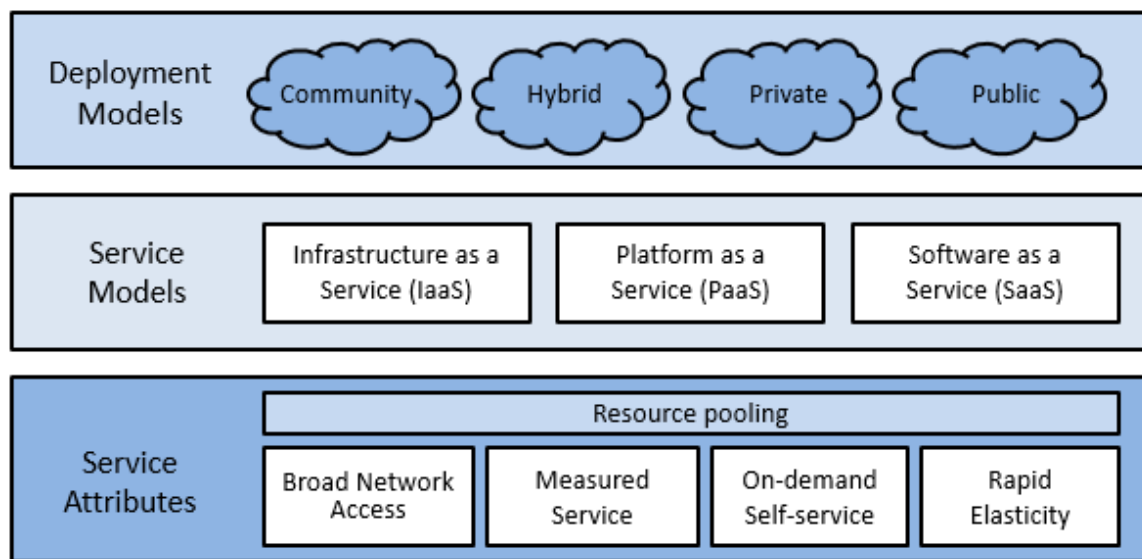


Figure 1.7: Cloud Computing - Source: (Clo 2014a)

#### 1.3.6.1 Software as a Service (SaaS)

In this model the capability provided to the consumer is to make use of the provider's applications operating on a cloud infrastructure. The requests come from different client devices either through a thin customer interface, such as a web browser (e.g., web-based e-mail), or a program interface. The customer does not manage or regulate the underlying cloud infrastructure, including network, servers, operating systems, storage, and even specific application capabilities, with the possible exception of restricted user-specific application setup settings. Examples of SaaS model are Abiquo's (Abi 2014) and Akamai (Aka 2014).



### 1.3.6.2 Platform as a Service (PaaS)

The capability offered to the customer is to deploy onto the cloud infrastructure, consumer-created or acquired applications produced utilizing programming languages collections, services, and devices supported by the provider. The customer does not handle or control the underlying cloud infrastructure, including network, servers, operating systems, or storage space, however, the customer has control over the deployed applications and potential configuration settings for the application-hosting environment. Examples of PaaS model are Heroku (Her 2014), EngineYard (Eng 2014), App42 PaaS (App 2014), and OpenShift (Ope 2014).

### 1.3.6.3 Infrastructure as a Service (IaaS)

The capability offered to the consumer includes providing processing, storage space, networks, and various other essential computing resources where the customer is able to deploy and run arbitrary software, which can consist of running systems and applications. The customer does not handle or control the underlying cloud infrastructure, but has control over operating systems, storage, and deployed applications; and potentially restricted control of selected networking components (e.g., host firewalls, etc.). Examples of IaaS model are Amazon Web Services (AWS) (Ama 2014), BlueLock (Blu 2014), Cloudscaling (Clo 2014b), and Datapipe (Dat 2014).

## 1.4 Comparison of Distributed and Parallel Systems

Parallel processing and distributed processing are closely related. In some cases, certain distributed techniques are used to achieve parallelism. As the communication technology advances progressively, the distinction between parallel and distributed processing becomes smaller and smaller.

Clusters are different from supercomputers; supercomputers are well adapted to solve large problems. Its hardware and maintenance cost is expensive thus only large organizations can afford to have it. Clusters are less expensive, easy to maintain therefore small organizations can afford it.

Distributed computing is different from Grid computing. Distributed system is a virtual computer formed by a networked set of heterogeneous/homogenous machines that agree to share their local resources with each other, whereas Grid is a very large scale generalized distributed system that can scale to Internet-size environments with machines distributed across multiple organizations and administrative domains with the involvement of more central resources.

The distinction between clusters and Grids is based on the resources that are managed by them. Clusters are owned by a single organization and resource allocation is performed by a centralized resource manager; nodes cooperatively work together as a single unified resource and connected over local area networks. Grids are an aggregation of clusters/server/supercomputer/personal computer, etc. Each node has its own resource manager, and does not aim at providing a single system view. Grids span multiple administrative domains and are connected over wide area network and are heterogeneous in nature.

Peer-to-Peer computing and Grid computing are concerned with sharing. Peer-to-peer is a collection of low end machines, and it allows sharing files, transferring money, voice communication, etc. Grids are a collection of high end machines (supercomputer, cluster) and personal computers, and it harnesses the storage, data, network within a virtual organization.

The distinction between Cloud and Grid computing is based on the ownership of resources. Cloud resources are owned by an industry or academic organization while Grids are mostly owned by academic organization. Clouds are liable to provide seamless services, whereas the Grid harnesses the underutilized resources. Table 1.2 illustrates different types of distributed computing systems.

## 1.5 Resource Management in Cluster Computing

Resource management is an integral part of cluster computing. There are various kinds of cluster management systems like centralized, decentralized, load balancing, and load sharing. Resource management system maximizes the system throughput and utilizes resources in a better way. Some widely developed, distributed resource management

Table 1.2: Different Types of Distributed Computing Systems

Parameters	Cluster	P2P	Grid	Cloud
System	Beowulf, Sun cluster and Windows cluster	Bittorrent, Skype and Bitvault	SETI@home Project and Large Hadron Collider	BlueLock, Cloudscaling and Datapipe
Architecture	Centralized	P2P	Decentralized	Dynamic Infrastructure
Application	Educational resources, Commercial sectors for industrial promotion, Medical re-search	MP3 File Sharing with Napster, Distributed Computing using SETI@Home, Instant Messaging with ICQ, File Sharing with Gnutella	Predictive Modeling and Simulations, Engineering Design and Automation, Energy Resources Exploration, Medical, Military and Basic Research, Visualization	Banking, Insurance, Weather Forecasting, Space Exploration
Network	LAN	LAN, WAN	LAN, MAN, WAN	MAN, WAN
Resources	More than 2 computers are connected to solve a problem.	Large number of non-dedicated computers are used to share resources.	A large project is divided among multiple computers to make use of their resources.	It does just the opposite. It allows multiple smaller applications to run at the same time.

systems are Condor, Maui, Portable batch schedule, and Load leveler that provide high performance. The next section describes some popular resource management systems for cluster computing.

### 1.5.1 Condor

Condor is a high-throughput resource management system that manages a heterogeneous pool of resources (Thain *et al.* 2005). It harnesses the computing power of idle resources by stealing the idle CPU cycles. Condor system follows a layered architecture. A set of resources managed by Condor is known as a *condor pool*. Condor keeps all the jobs submitted by the user in a queue. These jobs are then scheduled onto the machines in the pool, transparent to the user. Condor migrates a running job from one machine to another machine if, given machine fails to complete the given job.

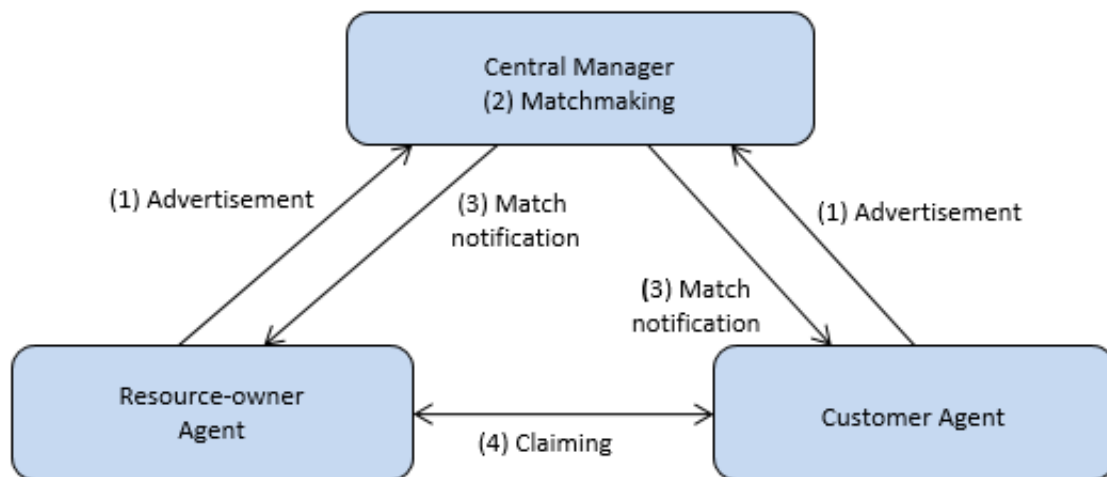


Figure 1.8: Condor Matchmaking Process - Source: (Thain *et al.* 2005)

Condor uses a resource specification language, known as Classified Advertisement language (ClassAds) to specify the resource requests. ClassAds uses a semi-structured data model and a query language as part of a data model that enables the advertising agents to include constraints to resource requests and offers and hence specifies their compatibility.

Condor has a centralized scheduling model and uses a dedicated machine, known as Central Manager, which is responsible for scheduling the jobs onto the resources in the condor pool. Condor matchmaking process (Thain *et al.* 2005) requires four steps as

shown in Fig. 1.8. In the first step, customer and resources agent advertise their characteristics and requirements in ClassAds. In the second step, a matchmaker scans the known ClassAds and creates pairs that satisfy each others constraints and preferences. In the third step, the matchmaker informs both parties of the match. The responsibility of the matchmaker then ceases with respect to the match. In the final step, claiming, the matched agent and resource establish contact, possibly negotiate further terms, and then cooperate to execute a job. The clean separation of the claiming step allows the resource and agent to independently verify the match.

Condor also supports preemption of jobs. In case a resource is withdrawn, then already running jobs are check-pointed and preempted to other resources, thus ensuring the resource owner autonomy as well as in-time completion of the jobs.

### 1.5.2 Portable Batch System

The Portable Batch System (PBS) (Yan & Chapman 2005) is designed to manage large parallel batch jobs running on multiple compute servers. It is based on a client-server paradigm. Clients make requests to the server to perform actions on a set of objects. Scheduler periodically collects the information about the jobs that are ready to run or currently running from the batch server.

PBS consists of four major components: commands, job server, job executor, and job scheduler. PBS provides commands and graphical interface to submit, monitor, modify, and delete jobs. Job server function is to provide the basic batch services such as receiving or creating a batch job, modifying the job, protecting the job against system crashes, and running the job (placing it into execution). Job executor is a daemon that places the job into execution by communicating Machine Oriented Miniserver (MOM). Job scheduler is another daemon, which contains the site's policy controlling (Loa 2014) which job to run and where and when to run it. PBS job scheduling and selection process are depicted in Fig. 1.9.

PBS includes several built-in schedulers, each of which can be customized for the local site requirements. Schedulers included in the suite are FIFO, Shortest Job First (SJF), and Fair Share. PBS server defines various kinds of queues for batch jobs like very long queue,

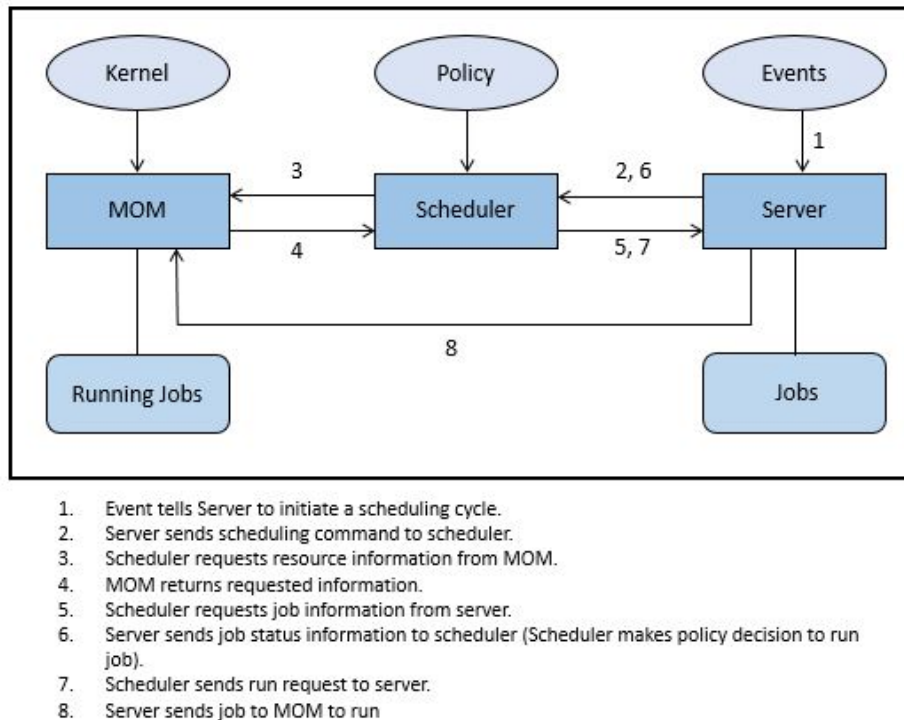


Figure 1.9: Portable Batch System - Source: (Yan & Chapman 2005)

short queue, and default queue. PBS maximizes the CPU utilization thus, it loops through the queued job list and starts any job which seems fit with the available resources.

### 1.5.3 Load Leveler

Load Leveler (LL) (Loa 2014) was developed to exploit CPU cycle stealing on workstations. LL does not follow centralized server configuration. User can submit and query jobs from any machine and then requested machine will send this request to the load negotiator. Load negotiator runs on a machine that is responsible for getting resources information and performs system wide task scheduling. Scheduling algorithms supported by LL are FIFO, Gang, and Backfilling.

### 1.5.4 Load Sharing Facility

Load Sharing Facility (LSF) (Pla 2005) is a suite of application resource management products that schedule, monitor, and analyze the workload for a network of computers. LSF supports sequential and parallel applications running as interactive and batch jobs. LSF is a loosely coupled cluster solution for heterogeneous systems that supports a num-

ber of scheduling mechanisms. There are several (like Fair share, Backfilling, and High Throughput) scheduling strategies available in LSF for managing priorities and deadlines. LSF scheduler performs load balancing and job migration among nodes in a cluster in case of load imbalance in the system.

### 1.5.5 Maui

Maui (Jackson *et al.* 2001) is an enhanced open source task scheduler that can team up with LSF and LL. Its emphasis is on rapid turnaround time of parallel jobs in a heterogeneous HPC environment.

The key to the Maui scheduler is its wall-time based reservation system. This system orders the queued jobs based upon priority, starts all the high priority jobs that it can, and then makes a reservation in the future for the next high priority job. As soon as this is done, the backfill mechanism attempts to locate lesser priority jobs that will fit into time gaps in the reservation system. This gives guaranteed start time to large jobs, while providing a quick turnaround time for smaller jobs. Maui is capable of supporting multiple scheduling policies, dynamic priorities, reservations, and fair share capabilities. The task of the job scheduler is to route the activities of the source manager, indicating when, where, and how tasks are to be started, preempted, and called off. It is also responsible for coordinating actions with other systems such as a Grid scheduler, allocation manager, or information service.

## 1.6 Resource Management in Grid Computing

A Grid resource management system is a middleware (Joshy & Craig 2003) that provides a cohesive and interoperable software solution. Grid middleware topology is shown in Fig. 1.10. A major component of middleware is security, resource management, information provider, and data management. The middleware provides security through integration of heterogeneous resources. Grid provides Grid Security Infrastructure (GSI) for single sign-on and platform integration. Due to a large number of heterogeneous resources, Grid resource management plays a crucial role in resources discovery, resources monitoring, job management, and resources selection. The most interesting aspect of the resource

management is the selection of the correct resources from the Grid resource pool, based on the service-level requirements, and then to efficiently provision them to facilitate user needs. Grid Resource Allocation and Management (GRAM) middleware provides this facility. Grid Information Service (GIS) provides static and dynamic information about resources. This includes resource utilization, availability, and capacity. This information is used by GRAM for resource management. The current advances in this platform are on virtualized data storage mechanisms, such as Storage Area Networks (SAN), Network File Systems (NFS), Dedicated Storage Servers (DSS), and Virtual Database (VD), etc. Following sections describe popular resource management systems of Grid computing.

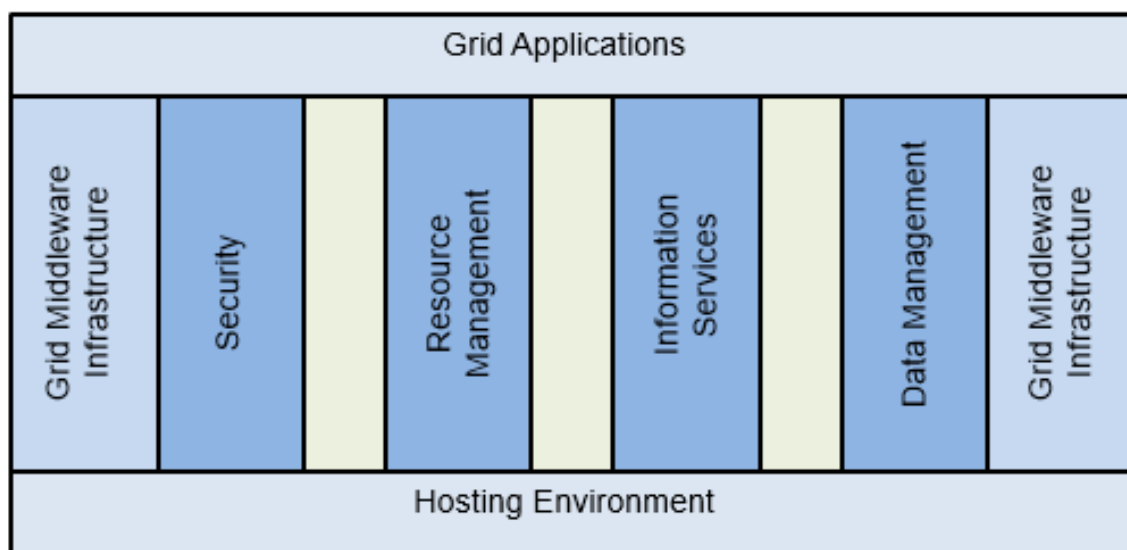


Figure 1.10: Grid Resource Management Topology - Source: (Joshy & Craig 2003)

### 1.6.1 Gridbus

Gridbus broker is a solitary user broker that sustains accessibility to both computational and information Grids (Gri 2005). It transparently connects with several types of computational resources, which are subjected by various community Grid middlewares such as Globus (Foster & Kesselman 1997), Unicore (Almond & Snelling 1999), and Amazon EC2 (Ama 2014), and organizing systems such as PBS and Condor. By nonpayment, it carries out two scheduling techniques that take into account budget plan and deadline of applications. Furthermore, the style of the broker enables the combination of customized organizing algorithms.



### 1.6.2 NetSolve

NetSolve (Seymour *et al.* 2005) is a client server system that enables user to perform computations remotely. The system allows users of FORTRAN, C, Matlab, Mathematica, Octave, or Excel to access both hardware and software computational resources distributed across the Grid. A NetSolve agent searches for computational resources, chooses the best available resource, retries for fault tolerance, and performs the computation. NetSolve uses a load-balancing policy to achieve good performance.

### 1.6.3 Legion

Legion (Chapin *et al.* 1999) is an object-based meta system that provides the software infrastructure for a Grid. In a Legion-based Grid, objects represent the different components of the Grid. Legion objects are defined and managed by the corresponding class or meta class. Classes create new instances, schedule them for execution, activate or deactivate the object, and provide state information to client objects. Each object is an active process that responds to method invocations from other objects within the system. Legion defines an Application Programming Interface (API) for object interaction, but does not specify the programming language or communication protocol. Although Legion appears as a complete vertically integrated system, its architecture follows the hierarchical model with Legion Class at the top and the host and vault classes at the bottom as shown in Fig. 1.11. It supports a mechanism to manage the load on hosts. It provides resource reservation capability and the capacity for application level schedulers to perform regular or batch scheduling. Legion machine architecture is hierarchical with a decentralized scheduler. Legion provides default system oriented scheduling policies; however, it enables policy extensibility through a structured scheduling extension interface.

### 1.6.4 Condor-G

Condor-G (Thain *et al.* 2005) is a fault tolerant system that can access different computers, which employs software applications from Globus and Condor to allocate resources to users in several domains. Condor-G has a task supervisor; therefore, it does not sustain scheduling plans. However, it provides a structure to execute scheduling designs con-

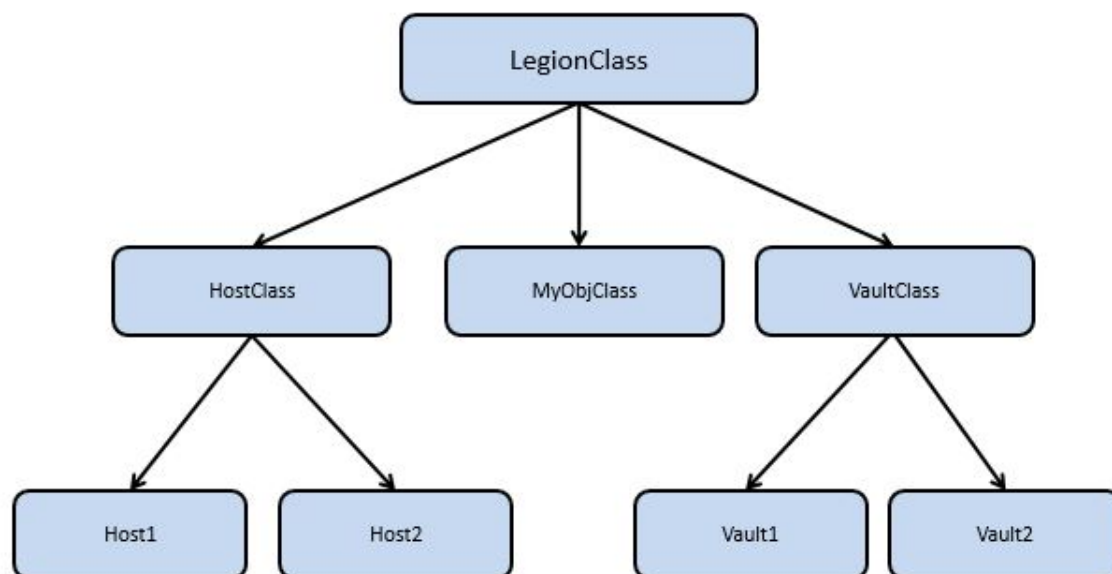


Figure 1.11: Legion Resource Management - Source: (Chapin *et al.* 1999)

cerning it. Condor-G can work together with the complying middleware: Globus Toolkit, Unicore and NorduGrid, and it can submit tasks to Condor, PBS and Grid Engine scheduling systems. Condor's ClassAd match making tool enables individuals to point out which resource to allot. The system permits both jobs and devices to explain attributes regarding themselves, their demands and inclinations, and matches an outcome in a logical-to-physical binding.

The Glidein (Sfiligoi 2008) system is, likewise, supplied in Condor-G that begins daemon processes, which could promote resource accessibility, which is used by Condor-G to match queued jobs to sources advertised. The command-line interface is supplied to carry out fundamental job management, such as sending a job, indicating executable input and output documents and disagreements, querying a job condition or revoking a task.

### 1.6.5 Nimrod-G

Nimrod-G is (Nir 2014) an automated and specialized source administration system, which permits implementation of parameter sweep applications on Grid to scientists and other types of users. Nimrod-G generally follows the commodity market model and provides four budget and deadline based algorithms for computationally-intensive applications. Each resource provider provides resources to the users. The users can vary

their QoS need based upon their requirement. Nimrod-G includes a Task Farming Engine (TFE) for creating and/or plugging user-defined scheduling policies and/or customised task farming applications. The task farming engine coordinates resource trading, scheduling, staging data and executable, execution, and gathers results from remote Grid nodes to the user home transparently. Nimrod-G is widely utilized in the areas of bio-informatics (Cab 2014).

### 1.6.6 Askalon

Askalon (Fahringer *et al.* 2005) is a Grid middleware for application development and computing environment whose goal is to provide transparent services to the application developers. Askalon provides four tools to the user: Scalea, Zenturio, Aksum, and Performance Prophet. Unlike other middleware systems such as Condor-G and Nimrod-G, Askalon is designed as a set of distributed Grid services using web services. Askalon supports workflow applications. A workflow application can be modeled as a Directed Acyclic Graph (DAG) where the tasks are the nodes and the dependencies between tasks are the arcs among the nodes. The user can describe workflows using the XML-based Abstract Grid Workflow Language (AGWL). Grid Askalon Resource Manager (GridARM), provides user authorization, resource management, resource discovery, and advanced reservation. Resource discovery and matching are performed based on constraints provided by the scheduler.

Askalon scheduler has a centralized architecture. It has three main components, as illustrated in Fig. 1.12: Workflow converter, Scheduling engine, and Event generator. Workflow converter converts AGWL into simple Directed Acyclic Graphs. Scheduler engine uses GridARM to get information about the Grid resources and maps the workflow onto resources using scheduling algorithm like Genetic Algorithm, HEFT, or Myopic based on user-defined QoS parameters. After the initial scheduling, the workflow is executed based on the current mapping until the execution finishes or any event is interrupted. Event generator module uses the performance analysis service to monitor the workflow execution and detect whether any of the contracts have been violated. In case of contract violation or if machine crashes, it performs a reschedule if necessary.

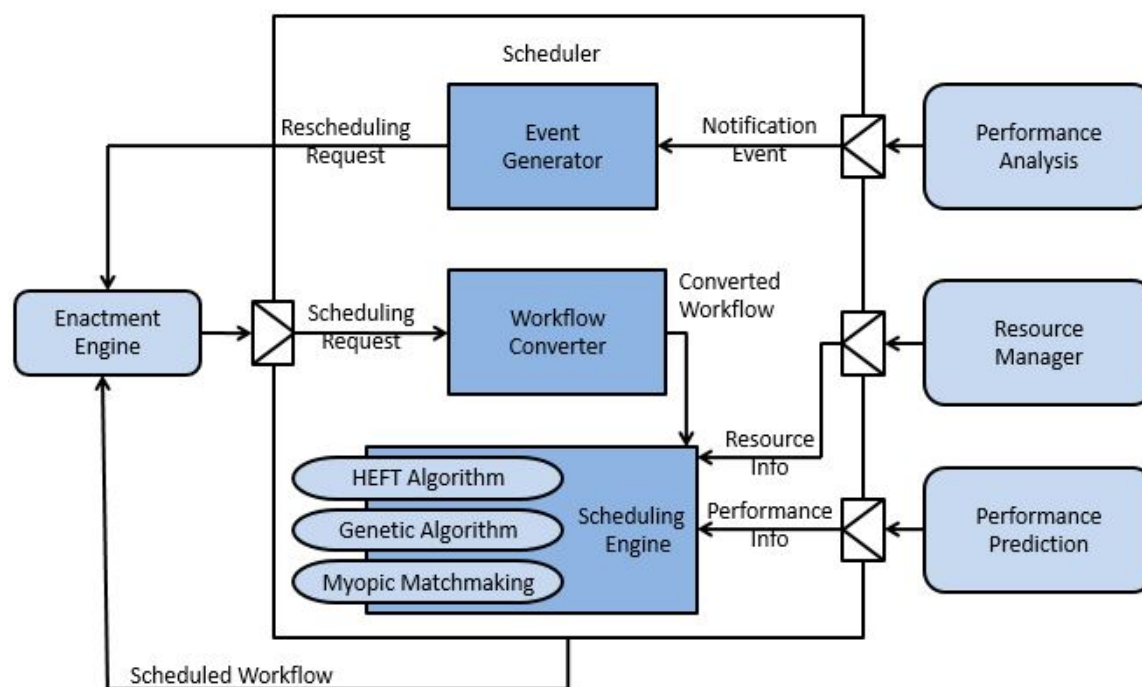


Figure 1.12: Askalon Scheduler Architecture - Source: (Fahringer *et al.* 2005)

### 1.6.7 Pegasus

Pegasus (Deelman *et al.* 2005) is part of the GridPhyN (Zhao *et al.* 2006) project and is a system that maps complex scientific workflows onto Grid resources. Pegasus uses GRAM (Foster & Kesselman 1997) for remote job submission and management; Monitoring and Discovery Service (MDS) to get information about the state of resources; Replica Location Service (RLS) to get information about the data available at the resource. Fig. 1.13 depicts the steps taken by Pegasus during the workflow refinement process.

Pegasus uses Directed Acyclic Graph Manager (DAGMan) and Condor-G to submit jobs on Globus-based resources. There are two main components in Pegasus: Pegasus Workflow Mapping Engine (PWME) and DAGMan workflow executor for Condor-G. PWME receives an abstract workflow description and generates an optimized concrete workflow. An abstract workflow describes the computation in terms of logical files and logical transformations and indicates the dependencies between the workflow components that can be described using Chimera's Virtual Data Language (VDL). A concrete workflow is an executable workflow that DAGMan can process. First, Pegasus queries the MDS to get information about the availability of the resources. The next step consists

of reducing the workflow to only contain the necessary tasks for the final product. This is done by querying the RLS for replicas of the required data. Next, Pegasus queries the Transformation Catalog (TC) to find the location of the logical transformation (software components) defined on the workflow. The obtained information is used to make scheduling decisions of various scheduling algorithms like Random-selection, Round-robin, and Min-min. It is also possible to add new scheduling algorithms to Pegasus.

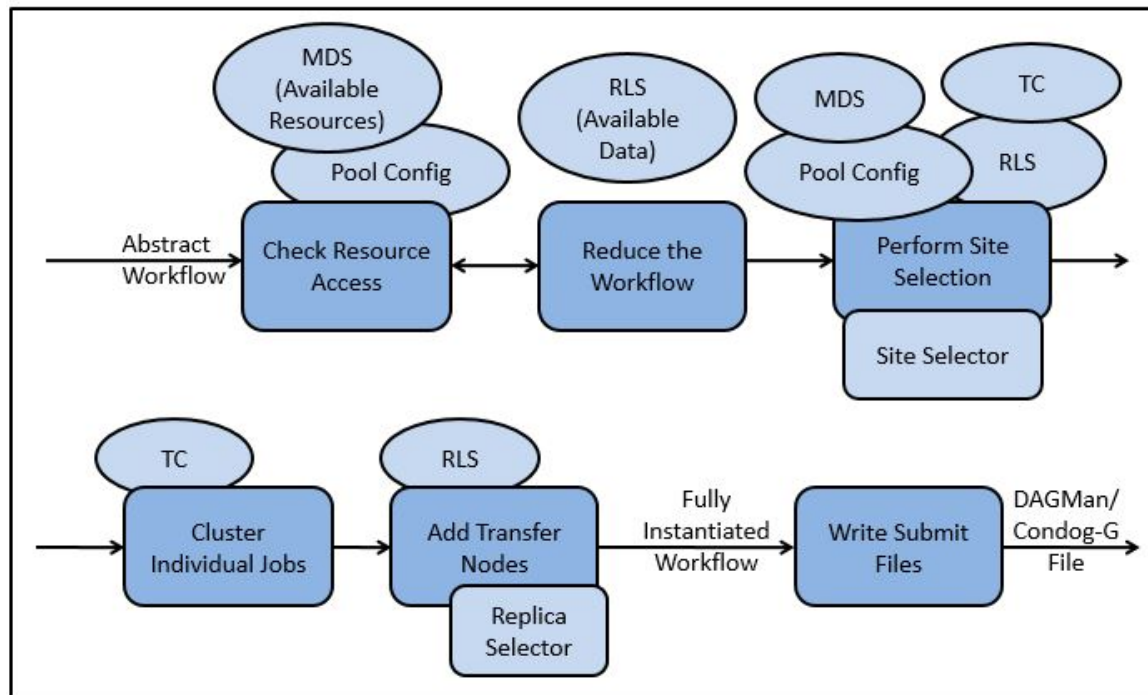


Figure 1.13: Pegasus Scheduler Architecture - Source: (Deelman *et al.* 2005)

## 1.7 Simulation Tools

Evaluation and comparative analysis of Grid scheduling algorithms are often difficult to perform. There are many causes; for example, difficulties in obtaining exclusive access to large scale infrastructures for research purposes or lack of certain functionality of real resource management systems, such as Advance Reservation (AR) or Grid user accounting. Therefore, Grid scheduling algorithms have been often tested in simulation environments. Simulators are useful to observe with high precision a local or global characteristic of a distributed system. The advantage of the simulators are their independence to the execution platform. Simulating large numbers of nodes of distributed system on a single PC

**Table 1.3:** Simulator Classification

<b>Simulators</b>	<b>Programming Language</b>	<b>Network Topologies</b>	<b>Workload Archive</b>
Bricks	JAVA	Yes	No
SimGrid	C/JAVA/Ruby	Yes	No
GridSim	JAVA	Yes	Yes
GSSIM	JAVA	Yes	No

is not rare. This advantage is made possible because the simulator does not run the real distributed system, but is a model of it. Key features of different simulators are tabulated in Table 1.3. The rest of this section describes some of the most popular Grid simulators.

### 1.7.1 Bricks

Bricks (Takefusa *et al.* 1999) was the first proposed Grid simulator designed for scheduling issues. Bricks was proposed and designed for studies and comparisons of scheduling algorithms and frameworks, under different structural and workload conditions. Bricks allows the simulation of diverse behaviors: resource scheduling algorithms, programming modules for scheduling, network topology of clients and servers in global computing systems, and processing schemes for networks and servers. It is basically a Java discrete event driven simulator where users can specify network topologies, server architectures, communication models, and scheduling framework components. It is possible to add new scheduling features by modifying a module called scheduling unit. Bricks has been in use for experiences associated with the Network Weather Service (NWS) for High-Energy Physics (HEP).

### 1.7.2 SimGrid

SimGrid (Legrand *et al.* 2003) was developed to study single-client multi-servers scheduling in the context of complex, distributed, dynamic, and heterogeneous environments. SimGrid is based on event driven simulation. Resources have characteristics like speed, availability, latency, and service rate. It provides a set of abstractions and functionalities to build a simulation corresponding to the applications and infrastructure characteristics. These characteristics may be set as constants or evolved according to previously collected traces. The topology is fully configurable and jobs have a cost and a state associated with

them. SimGrid is available in C, JAVA, and recently in Ruby (Sim 2014).

### 1.7.3 GridSim

The GridSim (Gri 2014) toolkit allows modeling and simulation of entities in parallel and distributed computing systems like users, applications, resources, and resource brokers for design and evaluation of scheduling algorithms. It provides a comprehensive facility for creating different classes of heterogeneous resources that can be aggregated using resource brokers, for solving computational and data intensive applications. A resource can be a single processor or multi-processor with shared or distributed memory and managed by time or space shared schedulers. The processing nodes within a resource can be heterogeneous in terms of processing capability, configuration, and availability. The resource brokers use scheduling algorithms or policies for mapping jobs to resources to optimize system or user objectives depending on their goals.

### 1.7.4 Grid Scheduling Simulator (GSSIM)

GSSIM (Kurowski *et al.* 2007) is a Java based discrete event simulator based on GridSim. GSSIM supports multilevel scheduling architectures with plugged-in algorithms both for Grid and local schedulers. It also enables both reading existing real workloads and generating synthetic Grid workloads based on given probabilistic distributions and constraints. These workloads are compliant with known workload formats such as Standard Workload Format (SWF) and Grid Workload Format (GWF). The framework also supports generation of resource failures which may be useful in modeling realistic behavior of Grid environments.

## 1.8 Description of Problem

A Grid is a system of high diversity, which is rendered by various applications, middle-ware components, and resources. But from the point of view of functionality, we can still find a logical architecture of the task scheduling subsystem in Grid. Grid scheduling architecture generalize a scheduling process in the Grid into three stages: resource discovering and filtering, resource selecting and scheduling according to certain objectives, and

job submission (Buyya *et al.* 2002). As a study of scheduling algorithms is our primary concern here, we focus on the second step. Based on these observations, Fig. 1.14 depicts a model of Grid scheduling systems in which functional components are connected by two types of data flow: resource or application information flow and task or task scheduling command flow.

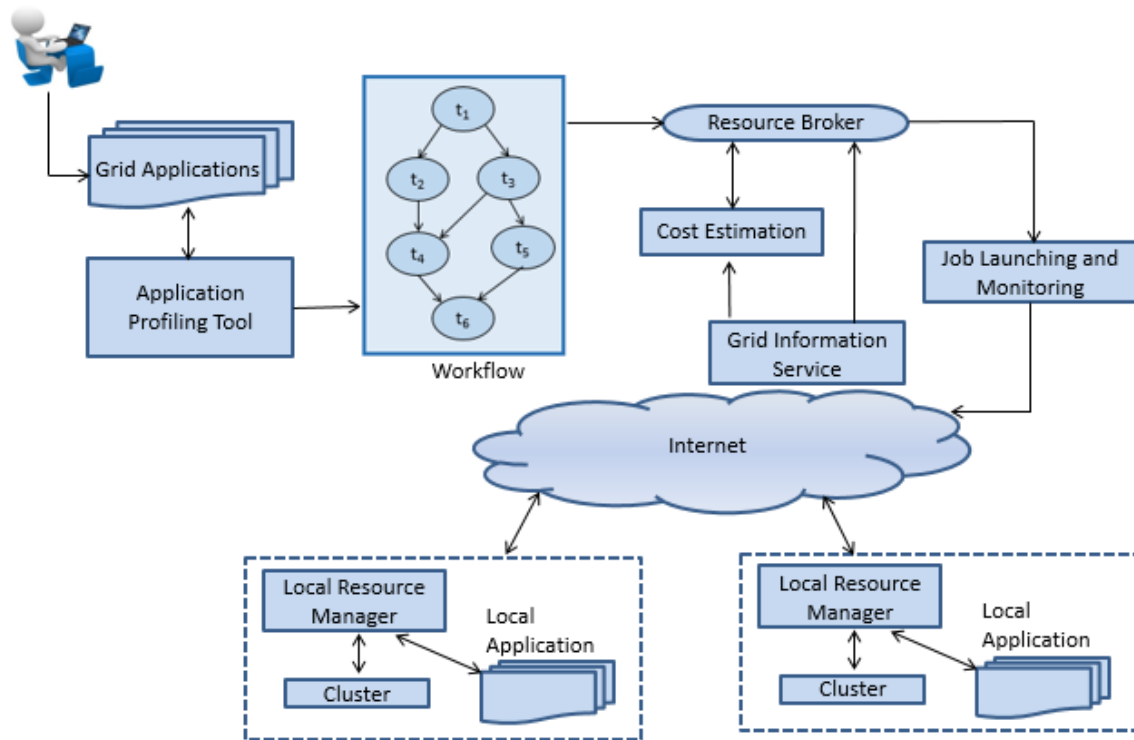


Figure 1.14: Grid Scheduling Architecture - Source: (Dong & Akl 2006)

Basically, a Grid scheduler (GS) receives applications from Grid users, selects feasible resources for these applications according to acquired information from the Grid Information Service module, and finally generates application-to-resource mappings, based on certain objective functions and predicted resource performance. Unlike their counterparts in traditional parallel and distributed systems, Grid schedulers usually cannot control Grid resources directly, but work like brokers or agents (Baker *et al.* 2002), or even tightly coupled with the applications as the application-level scheduling scheme proposes (Dong & Akl 2006). They are not necessarily located in the same domain with the resources which are visible to them. Fig. 1.14 only shows one Grid scheduler, but in reality multiple such schedulers might be deployed, and organized to form different structures (centralized, hierarchical, and decentralized) according to different concerns, such as per-



formance or scalability. Grid level scheduler/Metascheduler is crucial for harnessing the potential of Grids as they are expanding quickly, incorporating resources from supercomputers to desktops.

Information about the status of available resources is very important for a Grid scheduler to make a proper schedule, especially when the heterogeneous and dynamic nature of the Grid is taken into account. The role of the Grid Information Service (GIS) is to provide such information to Grid schedulers. GIS is responsible for collecting and predicting the resource state information, such as operating system, architecture and capacity, number of nodes in cluster, communication bandwidth, latency between clusters, and cluster current load. GIS can answer queries for resource information or push information to subscribers. The Globus Monitoring and Discovery System (MDS) (Foster & Kesselman 1997) is an example of GIS.

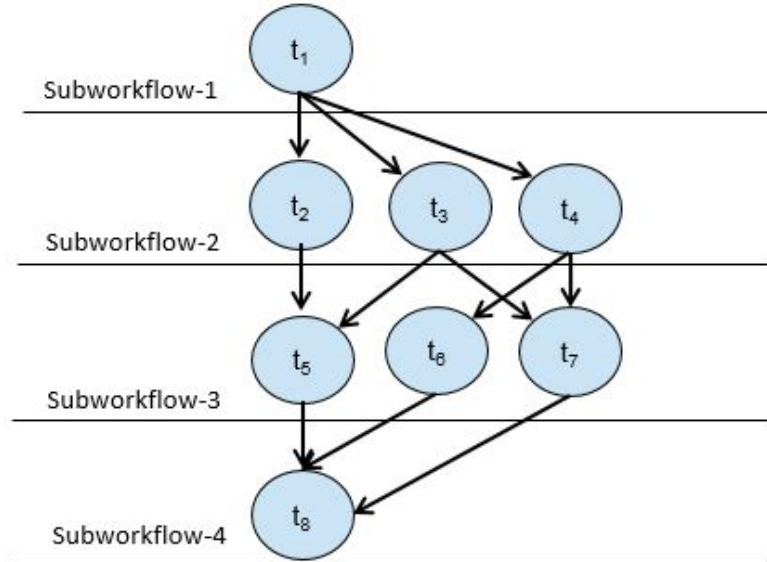
Besides raw resource information of resources it also requires application and performance of resources for different application. Application profiling tool (Hoschek *et al.* 2000) is used to extract properties of application. Application properties includes relationship (parent and child) among jobs in a workflow, number of jobs forming the workflow, size of each jobs in terms of the number of instructions and the number of bytes required to be exchanged between the two jobs in case of parent and child relationship between jobs. Analogical Benchmarking (Wolski *et al.* 1999), (Gong *et al.* 2002) provides a measure of how well a resource can perform a given type of application. On the basis of resource and application information and user specified criteria, resource broker maps applications to the resources. Criteria could be:

- Minimization of makespan, communication time, and cost of execution
- Satisfy the user defined QoS like deadline with budget
- Maximization of reliability, reputation, or security

The Launching and Monitoring module creates an agent to submit an application to selected resources, staging input output data and monitoring the execution of the applications.

A Local Resource Manager is responsible for local scheduling and reporting resource

information to GIS. Local scheduling schedules the local and Grid jobs. Example of local schedulers are PBS and Condor. Resource information is collected from Network Weather Service (Wolski *et al.* 1999) and report the resource status information to GIS.



**Figure 1.15: Workflow Partitioning Techniques** - Source: (Fahringer *et al.* 2005)

We have considered two scheduling models: Centralized and Decentralized. Centralized model schedules applications on multiple clusters that are located in multiple domain. In decentralized model schedulers interact among themselves in order to decide which resources should be allocated to the jobs being executed. In centralized model resource broker receives a Grid enabled application from user, and application is converted into Directed Acyclic Graph (Workflow). After that, resource broker either maps a full workflow (Dependent task scheduling algorithm) or subworkflow (Independent task scheduling algorithm). Dependent task scheduling algorithm considers parent and child relationship between tasks, a child task cannot start until its parent task completes execution. For example, in Fig. 1.15 task ( $t_3$ ) cannot start execution until task ( $t_1$ ) complete execution. Independent task scheduling is formed using Iterative workflow partitioning technique (Fahringer *et al.* 2005). Workflow partitioning technique schedules a workflow into a sequence of subworkflows, which are subsequently scheduled and executed. For example Fig. 1.15, has four subworkflows such that subworkflow-1, subworkflow-2, subworkflow-3, and subworkflow-4, have tasks ( $t_1$ ), ( $t_2, t_3, t_4$ ), ( $t_5, t_6, t_7$ ), and ( $t_8$ ) respectively. Task scheduling algorithm is further classified as online and offline. Online

scheduling algorithm schedules a tasks to clusters as they arrive into the system. Offline scheduling algorithm schedules the tasks to clusters after scheduling interval.

## 1.9 Thesis Contributions

In this thesis new Grid scheduling algorithms of various categories are designed, known as dependent task scheduling algorithm, independent task scheduling algorithm and distributed scheduling algorithm. The contributions are as follows:

- Dependent task scheduling algorithm is named as Double Hybrid Non Dominated Sorting Algorithm has been developed. This scheduling algorithm works on economic Grid. It minimizes communication cost, computation cost and makespan. Its results are compared with existing widely referred meta scheduling algorithm. The comparison is done on the real world problem, known as Gaussian elimination algorithm. Results show that Double Hybrid Non Dominated Sorting Algorithm outperformed the existing algorithm in terms of solution quality and hyper volume matrix. Further solutions are ranked because in multi-objective no solution is better than other solutions.
- Two novel independent task scheduling algorithms have been designed. Enhanced Refinery heuristic is worked on computational Grid. It minimizes makespan. It outperforms the counter heuristics, in terms of makespan. Parallel tasks scheduling algorithm schedules parallel tasks on economic Grid. The simulation results show that parallel scheduling algorithm performs better than existing approaches in terms of average cost, average makespan and number of failure.
- In the final part of this thesis, we explored two decentralized scheduling algorithms where nodes itself decide when and where to schedule newly arrived tasks. Enhanced Sender-initiated scheduling algorithm works in heterogeneous system, in which system threshold is changed dynamically. The results obtained using the proposed scheduling algorithm improves over the existing approaches in terms of number of messages and turnaround time. Efficient Dynamic Round Robin scheduling algorithm, models a scheduling system as a state-transition diagram and repli-

cates a task intuitively. This scheduling algorithm outperforms the other existing algorithms in terms of average response time.

## 1.10 Thesis Organization

The rest of the thesis is organized as follows. A brief introduction of distributed and parallel systems is given in Chapter 1. Survey of the current literature is presented in Chapter 2.

Chapter 3 presents dependent task scheduling algorithm on Grid. This scheduling algorithm minimizes three conflict objectives namely makespan, communication cost, and computation cost of execution using NSGA-II. Various version of NSGA-II has been tested and new Double Hybrid NSGA-II version is introduced. A detailed simulation study and its results are discussed to reveal the benefits attained and compromises reached as compared to the results obtained with the use of models based on above mentioned three objectives. The chapter ends with discussion on the model's performance.

Chapter 4 presents two independent task scheduling algorithms. Enhanced Refinery heuristic works on computational Grid that minimizes makespan. Parallel task scheduling algorithm works on economic Grid that minimizes makespan, cost, and processor fragmentation. A detailed simulation study and its results are discussed to reveal the benefits of proposed algorithms.

Chapter 5 presents two decentralized Grid scheduling algorithms, namely Efficient Dynamic Round Robin scheduling algorithm and Enhanced Sender-initiated scheduling algorithm. Efficient Dynamic Round Robin scheduling algorithm model a scheduling algorithm as a state transition diagram and duplication candidate task is chosen intuitively to avoid impractical duplication. Enhanced Sender-initiated scheduling algorithm works on Grid system where nodes have heterogeneous in nature and uses polling information to determine threshold.

Chapter 6 concludes the thesis by analyzing the various models presented with a comparative evaluation of their performance. This section throws light on the achievements made by the work and the areas where a further exploration is required along with the future directions that may be undertaken.

## Chapter 2

# State of the Art

Distributed systems offer a tremendous processing capacity. However, in order to realize this tremendous computing capacity, and to take full advantage of it, good resource allocation schemes are needed. A scheduler is a resource management component of a distributed system that focuses on judiciously and transparently redistributing the load of the system among the computers so that the overall performance of the system is maximized. In this chapter, we discuss various scheduling methods available in the distributed computing literature.

### 2.1 Scheduling Algorithms in Distributed Computing

Distributed scheduling algorithms achieve better system performance by smoothing out any workload imbalance that may exist in a distributed system, such as minimizing communication delays, minimizing execution time, and maximizing resource utilization, etc. Load scheduling is the process of deciding where to execute a process in a multi-computer system. This can be carried out by a single authority or by many entities spread in a distributed manner. It also decides whether to equalize the load at all the computers or to share the load between highly loaded to lightly loaded nodes.

Load distributing algorithms are classified as static, dynamic, and adaptive. In a static algorithm, the scheduling algorithm is carried out according to a predetermined policy. The state of the system at the time of the scheduling is not taken into consideration. On the other hand, a dynamic algorithm adapts its decision to the state of the system.

Adaptive algorithms are a special type of dynamic algorithms where the parameters of the algorithm are changed based on the current state of the system.

The policies adopted by a load balancing algorithm are as follows: Transfer policy decides when to initiate load balancing across the system and whether the node is a sender or a receiver. This is decided by a threshold that is called load index. Selection policy determines which task should be transferred, whether it be the newly arriving job or a job that has been executed for some time. Information policy specifies the information about the load level of a node that is made available to the job placement decision makers. System information can be collected periodically or by a demand driven approach or by a state change driven approach. Location policy determines the node to which a process has to be transferred, where the selection is made on the basis of load index of the node. Location based policies can be broadly classified as Sender-initiated, Receiver-initiated, and Symmetrically-initiated. Section 2.1.1 to 2.1.5 describes some popular distributed scheduling algorithms.

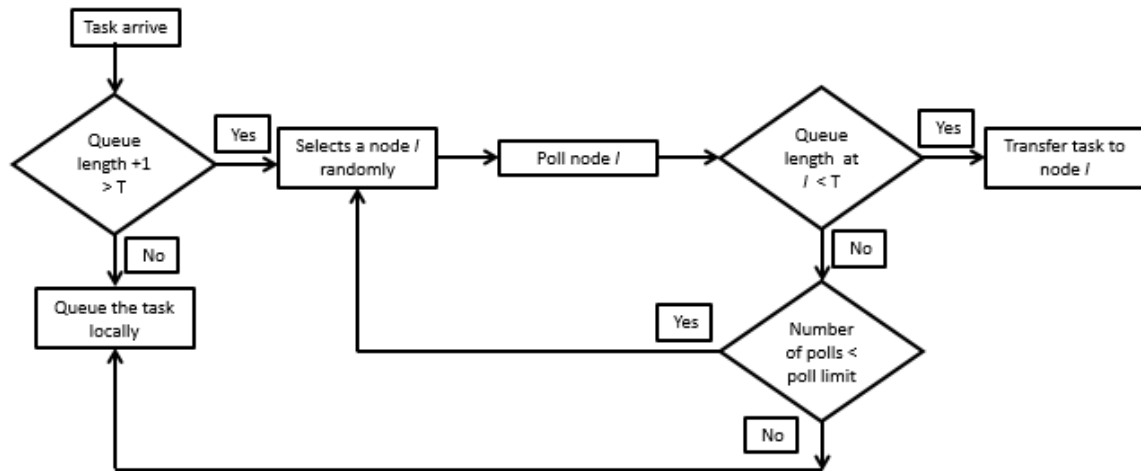
### 2.1.1 Sender-initiated Scheduling Algorithm

The working of Sender-initiated scheduling algorithm (Shivaratri *et al.* 1992) is shown in Fig. 2.1. In this algorithm whenever a new task arrives at a node, node computes its queue length. If the node's queue length+1 is greater than the threshold, node acts as a sender. Sender node randomly polls a node  $i$  in the system. If the polled node queue length is less than predefined threshold  $t$ , sender transfers the newly arrived task to the polled node. This process is repeated until, poll count is lesser than poll limit. Otherwise the task is processed locally.

Drawback of Sender-initiated algorithm is that it is not stable in high system load because polling activity increases the system load and wastes the CPU cycle.

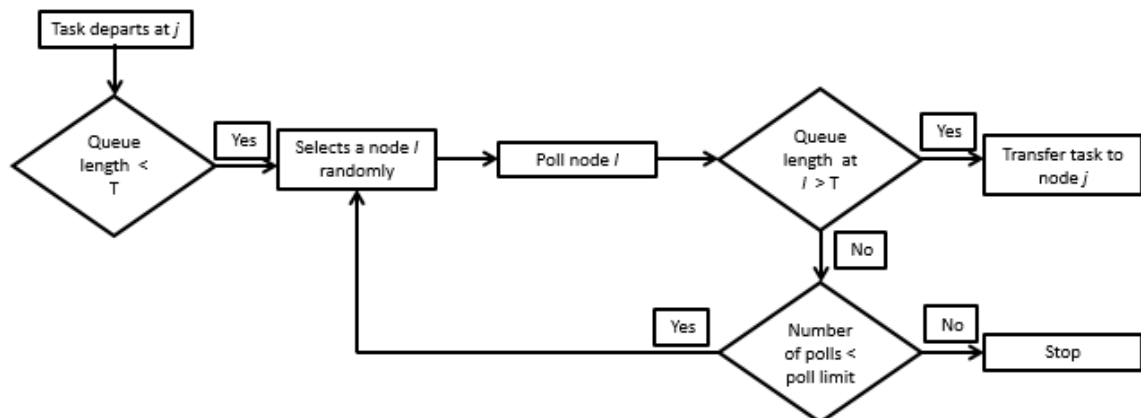
### 2.1.2 Receiver-initiated Scheduling Algorithm

This scheduling algorithm (Shivaratri *et al.* 1992) is similar to Sender-initiated scheduling algorithm. Instead of arrival a new task at a node, it works when a task departs from a node. The working of Receiver-initiated algorithm is shown in Fig. 2.2. In this algorithm



**Figure 2.1: Working of Sender-initiated Scheduling Algorithm** - Source: (Shivaratri *et al.* 1992)

whenever a task departs from a node, the node checks its the queue length. If the queue length is less than the predefined threshold  $t$ , node randomly polls the nodes  $i$  in the system, until either it gets the task from the overloaded node or reaches the poll limit.



**Figure 2.2: Working of Receiver-initiated Scheduling Algorithm** - Source: (Shivaratri *et al.* 1992)

This scheduling algorithm does not suffer from instability because in high system load there is low probability that node acts as a receiver. If any node acts as a receiver, it will find a sender within few polls, wherein, low system load polling activity does not increase the system load. The drawback of this algorithm is mostly preemptive transfer that is costly, because process's state needs to be transferred.

### 2.1.3 Symmetrically-initiated Scheduling Algorithm

Symmetrically-initiated scheduling algorithm (Shivaratri *et al.* 1992) is a combination of Sender-initiated and Receiver-initiated scheduling algorithms. At the high system load receiver finds the sender and at the low system load, sender finds the receiver. This algorithm does not remove the drawback of Sender-initiated and Receiver-initiated scheduling algorithms (Shivaratri *et al.* 1992) because in high system load sender polling activity increases the system instability and mostly receiver initiated transfers are preemptive.

### 2.1.4 Stable Symmetrically-initiated Scheduling Algorithm

Stable Symmetrically-initiated scheduling algorithm (Shivaratri *et al.* 1992) is an adaptive algorithm that stores the information, which has been collected during the polling, to classify the node in the system as sender, receiver, or okay. The state of nodes is maintained by a data structure at each node, comprises of sender, receiver, and okay lists. These lists are maintained by efficient data structure so that manipulation takes constant time. Initially, each node assumes that every other node is a receiver. The used policies are described as follows:

- Transfer policy: Queue length based transfer policy is used. It maintains two thresholds namely Lower Threshold (*LT*) and Upper Threshold (*UT*). Whenever a task departs/arrives at a node, node decrements/increments its Queue (*Q*) length respectively. After that, if the *Q* length is between the *LT* and *UT* node act as an okay node, else if *Q* length is less than the *LT* node acts as a receiver, otherwise node act as a sender.
- Selection policy: The sender-initiated component considers only newly arrived tasks for transfer, whereas receiver-initiated component uses preemptive transfer or reservation based policy.
- Information policy: The information policy is demand driven that is whenever a node becomes sender/receiver, polling activity starts.
- Location policy: The location policy has two components namely sender-initiated and receiver-initiated component.



- Sender-initiated component: It is triggered when a node becomes a sender. The sender polls the node at the head of the receivers list to determine whether it is still the receiver or not. The polled nodes updates the sender's node status in its own list and inform the sender about its status. On receiving the reply message the sender updates its list, if reply message is positive (still receiver), sender transfers the newly arrived task. Otherwise sender polls the next node from the receiver list.

This polling process stops if a suitable receiver is found for the newly arrived task, if the number of polls reach a poll limit, or if receiver list at the sender becomes empty. If polling fails to find a receiver the task is processed locally, and later it can migrate when receiver component is initiated at that node.

- Receiver-initiated component: This component is triggered when a node departs a task. The receiver polls the nodes in the following order: head to tail in the sender list, then tail to head in the okay list because most out of date information is used first, in the hope that node has become a sender, then tail to head in the receiver list so that the most out of date information is used first. The receiver polls the selected node to determine status. On receiving the reply message receiver updates its list, if the reply is positive (polled node is sender) receiver receives a task, otherwise polls to the next node from the list.

The polling process stops if a suitable sender is found, if the number of poll reaches the poll limit or the receiver is no longer a receiver.

Drawback: This algorithm has the same disadvantages that we have in Symmetrically-initiated scheduling algorithm. Even, it becomes worse at high system loads, because unsuccessful polls result in the removal of polled node ids from receivers lists. This scheme prevents future sender-initiated polls at high system loads and hence, the sender-initiated component is deactivated at high system loads, leaving only receiver-initiated load sharing which is effective at such loads. At low system loads, receiver-initiated polls are frequent and generally fail. These failures do not adversely affect performance, since extra processing capacity is available at low system loads. In addition, these polls have the positive effect of updating the receiver's lists. With the receivers lists accurately reflecting

the system's state, future sender-initiated load sharing will generally succeed within a few polls. Thus, by using sender-initiated load sharing at low system loads, receiver-initiated load sharing at high system loads, and symmetrically-initiated load sharing at moderate system loads, the Stable Symmetrically-initiated algorithm achieves improved performance over a wide range of system loads and preserves system stability.

### 2.1.5 Stable Sender-initiated Adaptive Scheduling Algorithm

Stable Sender-initiated Adaptive scheduling algorithm (Krueger & Finkel 1984) uses the sender-initiated load-sharing component of the previous approach, but modifies receiver-initiated component to attract future non preemptive task transfers from sender nodes. An important feature is that the algorithm performs load sharing only with non-preemptive transfers, which are cheaper than preemptive transfers. In the following section, we point out the differences. In this scheduling algorithm, the data structure (at each node) of the Stable Symmetrically-initiated scheduling algorithm is augmented by an array called the state vector. Each node uses the state vector to keep track of which list (senders, receivers, or okay) it belongs to at all the other nodes in the system. For example, state vector [node id] says, to which list node  $i$  belongs at the node indicated by node id. As in the Stable Symmetrically-initiated scheduling algorithm, the overhead for maintaining this data structure is small and constant because state vector is maintained in an array structure that takes constant time to access an item, irrespective of the number of nodes in the system.

The sender-initiated load sharing is augmented with the following steps: When a sender polls a selected node, the sender's state vector is updated to show that the sender now belongs to the senders list at the selected node. Likewise, the polled node updates its state vector based on the reply it sent to the sender node to reflect which list it will belong to at the sender.

The receiver-initiated component is replaced by the following protocol: When a node becomes a receiver, it informs only those nodes that are misinformed about its current state. The misinformed nodes are those nodes whose receivers lists do not contain the receiver id. This information is available in the state vector at the receiver. The state

vector at receiver is then updated to reflect that it now belongs to receiver at all those nodes. By this technique, this scheduling algorithm avoids receivers sending broadcast messages to inform other nodes that they are receivers. The broadcast messages impose message handling overhead at all nodes in the system. This overhead can be high if nodes frequently change their state.

## **2.2 Scheduling Algorithms in Parallel/Cluster Computing**

Multitasking operating systems are able to execute several tasks at once by allocating resources to different tasks for a very short period of time. This method is called time sharing scheduling because applications share the resources in turn. This strategy can be applied in clusters as well. However, using a time sharing scheduling policy with high performance parallel application, generally leads to very poor performance. Indeed, switching between applications has a cost, therefore it is usually better to execute applications one after the other. Furthermore, two applications with large memory requirements may not be able to run concurrently on a single machine of the cluster. For above mentioned reasons (Schwiegelshohn & Yahyapour 1998), schedulers in clusters usually use a space sharing policy where each application has a dedicated access to the resources for a given period.

Therefore, in order to execute a job on a cluster, users must submit their job to a batch scheduler that gives a dedicated access to the resources for some time. Batch schedulers keep a Gantt chart/diagram of the resources. X-axis of the diagram corresponds to time and Y-axis represents the processors/machines. When a job is submitted, the batch scheduler looks for a place where the application can be executed. Therefore, finding a schedule is equivalent to finding a tiling of this 2D plane, where the plane represents resource availability over time and tiles represent jobs. Thus, submissions of jobs must include a description of their requirements: the number of processors needed as well as their duration. Most schedulers are only able to schedule rigid tasks. Giving the duration of a job before its execution is usually impossible, therefore a wall time is provided. The wall time of a job is the expected duration of the job. Batch schedulers use this information to perform the scheduling. If the wall time is underestimated, jobs are usually killed,

so the users give an overestimation of the expected execution time. Some of the most commonly used algorithms in batch scheduling are described next.

**2.2.1 First-Come First-Served (FCFS) Scheduling Algorithm**

FCFS (Schwiegelshohn & Yahyapour 1998) is the simplest scheduling algorithm. This scheduling algorithm schedules the jobs in the order of their arrival. If enough resources are not available, job waits until enough resources are available. If a task finishes before its wall clock time, tasks are resubmitted in their submission order. Thus, a job cannot be delayed by a job that arrives after it, as shown in Fig. 2.3. In Fig. 2.3, job 1 and 2 are running, job 4 arrives after job 3. It is scheduled after job 3, even when the required number of processors are available. FCFS scheduling algorithm suffers from starvation problem when smaller tasks wait behind a longer task.

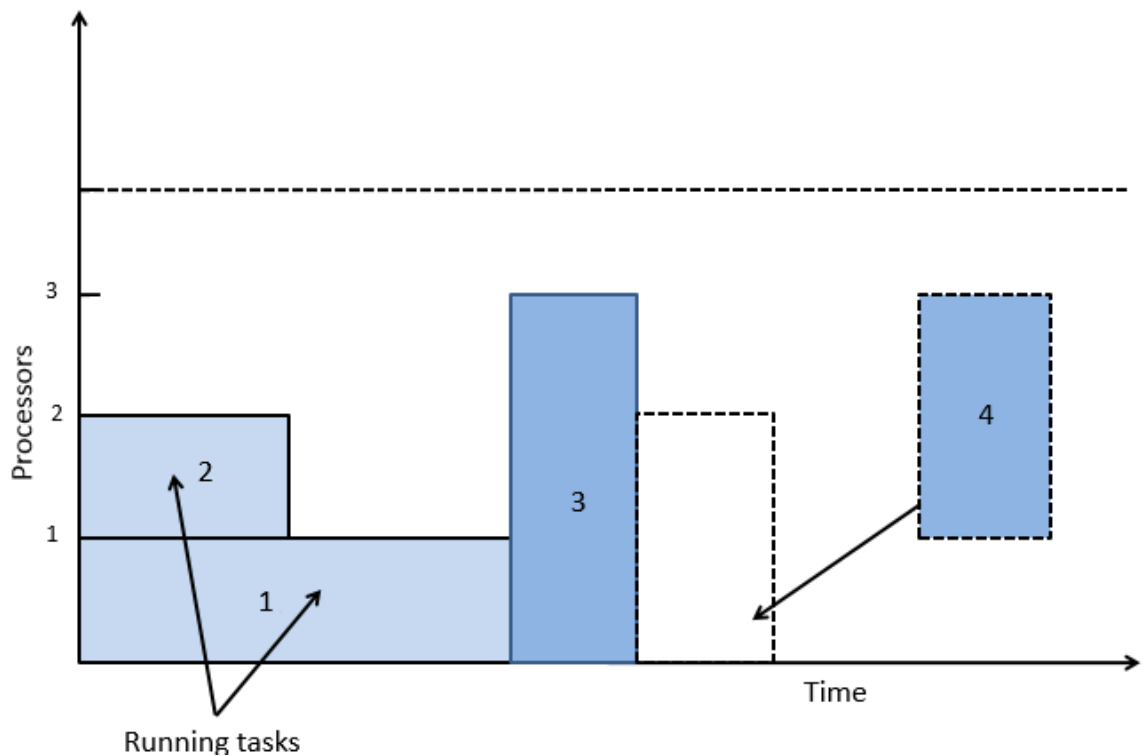


Figure 2.3: First-Come First-Served Example - Source: (Schwiegelshohn & Yahyapour 1998)

**2.2.2 Conservative Backfilling (CBF) Scheduling Algorithm**

CBF scheduling algorithm (Muallem & Feitelson 2001) removes the disadvantages of FCFS scheduling algorithm. CBF scheduling algorithm uses empty spaces present in the waiting

queue. In this system when a new job is submitted into the system, it goes to the front of the queue. If any space is found, it fills that space, with the objective that the scheduled jobs should not be delayed. Working of CBF scheduling algorithm is described in Fig. 2.4. In the Fig. 2.4, job 1 and 2 are running and job 4 arrives after job 3 that requires less number of processors, that are available for the required time, therefore, job 4 is scheduled before the job 3. This scheduling algorithm improves the resource utilization.

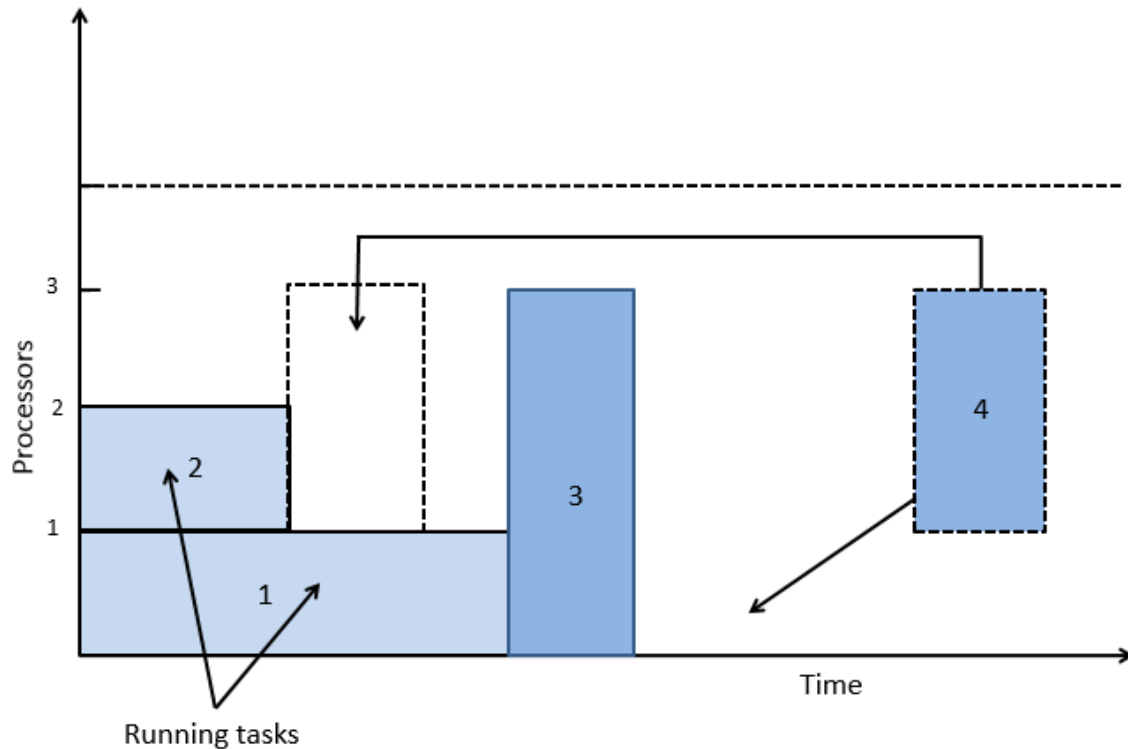


Figure 2.4: Conservative Backfilling Example - Source: (Mualem & Feitelson 2001)

### 2.2.3 Aggressive Backfilling (ABF) Scheduling Algorithm

ABF scheduling algorithm (Lifka 1995) is a variation of CBF scheduling algorithm. This scheduling algorithm allows the delay of scheduled jobs. Fig. 2.5, illustrates the process of ABF scheduling algorithm. Here, job 3 arrives after job 2 and required number of processors are available before the scheduled job 2, but not for required time. Still, it will be scheduled by delaying the job 1, which is acceptable in ABF scheduling algorithm. Thus, ABF scheduling algorithm does not give guaranty of the start time of a job and jobs are delayed for a long time and suffer from starvation problem.

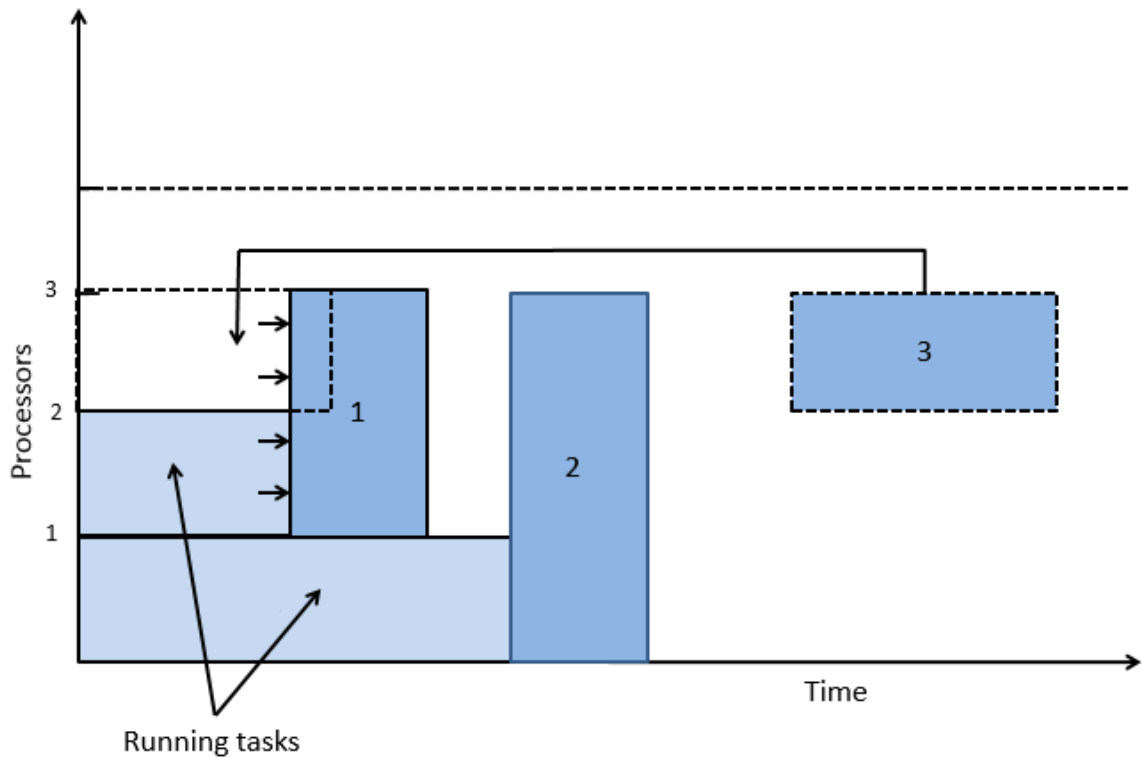


Figure 2.5: Aggressive Backfilling Example - Source: (Lifka 1995)

### 2.2.4 Easy Backfilling (EASY) Scheduling Algorithm

EASY scheduling algorithm (Feitelson *et al.* 2004) removes the starvation problem of ABF scheduling algorithm. This scheduling algorithm works similar to ABF scheduling algorithm except it does not allow the delay of running job as shown in Fig. 2.6. Job 3 is scheduled before the job 2 because required number of processors are available and job 2 can be delayed because it is not running. It improves the resource utilization of the system.

All the scheduling algorithms that are mentioned in section 2.2 work in an online manner. They schedule each job independently upon their submission. Other scheduling algorithms work in an offline manner such that, Backfilling with Look Ahead scheduling algorithm (Shmueli & Feitelson 2005a) uses packing techniques (Shmueli & Feitelson 2005b) to maximize the resource utilization at each job submission. The packing techniques use the knowledge of all the jobs each time. This kind of scheduling algorithm is not widely used because the execution time needed to execute the algorithm can easily become too long.

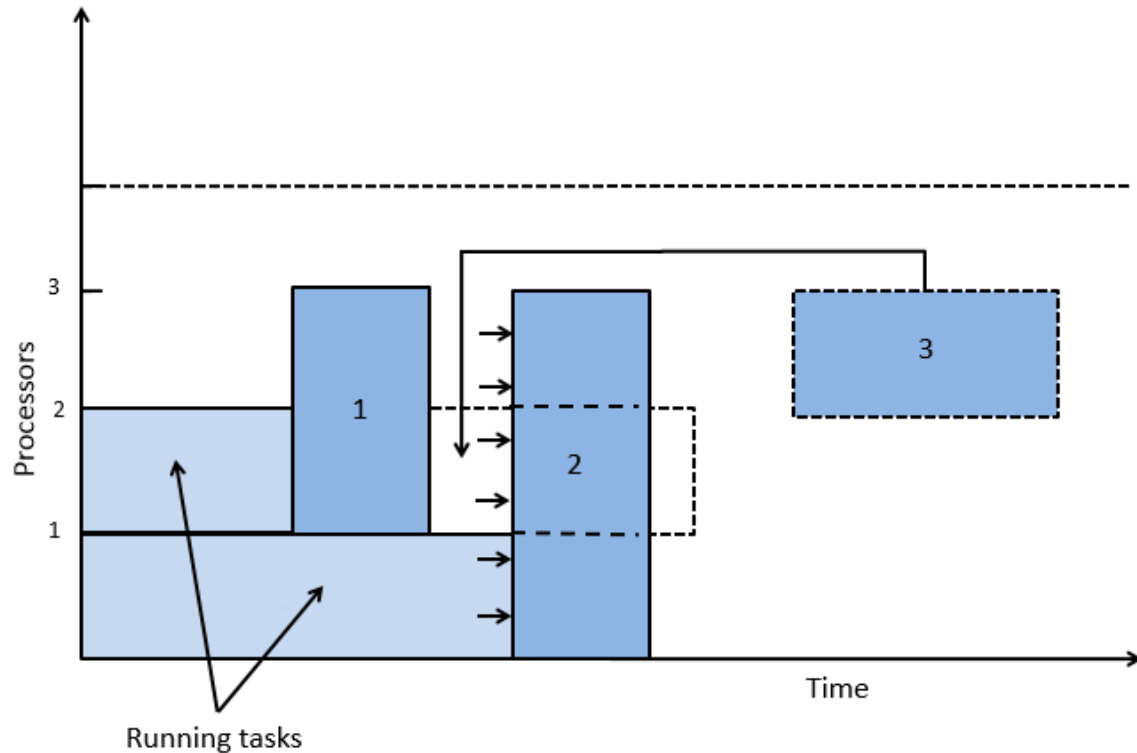


Figure 2.6: Easy Backfilling Example - Source: (Feitelson *et al.* 2004)

### 2.3 Scheduling Algorithms in Grid Computing

Grid is a kind of parallel and distributed computing that is not administered centrally. Therefore, scheduling jobs in a Grid is quite different than scheduling on a distributed computing. Indeed cluster computing is homogeneous and have a reasonable size. However, a Grid can embed different kinds of hardware, software, and scheduling policies. Furthermore, the number of resources available in a Grid is a lot larger than on distributed computing and resource access is done differently. Distributed computing is accessed through local resource management systems, but Grids are usually accessed through a resource broker. Job scheduling in a Grid, therefore, takes all these additional constraints into account.

Grid scheduling works in three phases, namely resource discovery and filtering, system selection and scheduling, and job launching and monitoring. Resource discovery and filtering phase performs authorization and gathers initial information about resources. System selection and scheduling phase collects information about application and resources, it designs a schedule based on objective functions or QoS constraints. Job launch-

ing and monitoring, prepares the job for transfer, transfers the jobs to local resources and monitors its execution. Local resources execute the tasks. Our work emphasizes on system selection and scheduling in Grid computing system.

### 2.3.1 Objective Functions

Task scheduling objective functions are classified into user and resources centric. User centric includes makespan or flow time. Resource centric includes resource utilization and system load balance. These objective functions (Xhafa & Abraham 2010) are shown in Fig. 2.7 and described in section 2.3.1.1 to 2.3.1.4.

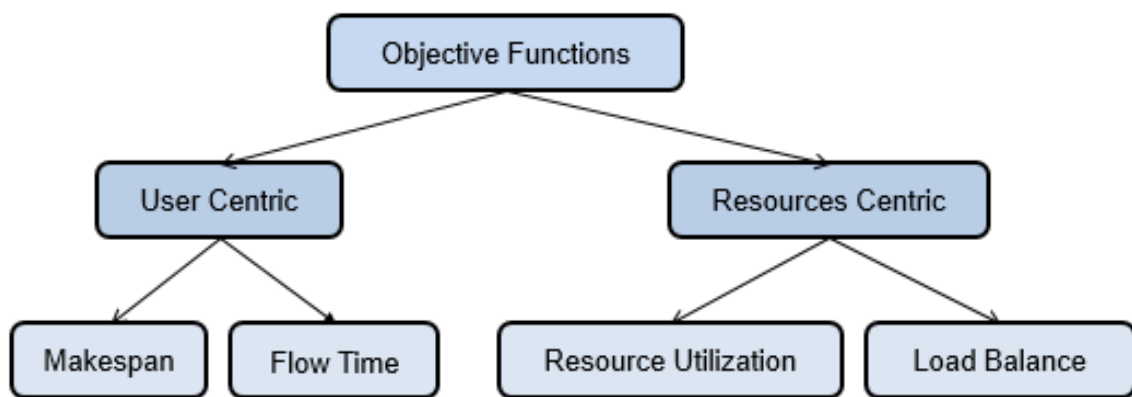


Figure 2.7: Classification of Objective Functions

#### 2.3.1.1 Makespan

Makespan corresponds to the total duration (overall job completion time) required to execute all the jobs of a schedule. It is the difference between the last job completion time and the first job start time. The scheduling heuristics that are based on the makespan, try to minimize it, so that all the computations are finished in minimum time. The makespan is influenced by the order in which jobs are executed. The completion time of job<sub>*i*</sub> is computed as:

$$ct_i = readytime_j + ETC[i][j] \quad (2.1)$$

here  $ct_i$  is the completion time of job<sub>*i*</sub>. It includes ready time and Expected Time to Compute (ETC) of machine<sub>*j*</sub>. Completion of machine<sub>*j*</sub> is defined as:



$$ct_j = \sum_{k \in K; m \in M} ct[k][j] \quad (2.2)$$

Here,  $ct_j$  is the sum of completion time of machine  $j$ ,  $M$  is the number of machines and  $K$  is the set of assigned tasks to machine  $m$ . Makespan defined as a function of  $ct$ . It is the maximum completion time of all machines and is given as follows:

$$makespan = \max(ct_j, \dots, ct_m) \text{ for } (j = 1, 2, \dots, m) \quad (2.3)$$

### 2.3.1.2 Flow Time

Flow time is the sum of the finishing times of jobs. Flow time is minimum, when jobs are processed in ascending order of processing time on resources. It is defined as follows:

$$flowtime = \sum_{i=0}^n ct_i \quad (2.4)$$

Here,  $n$  is the number of jobs.

### 2.3.1.3 Average Resource Utilization Rate

Average resource utilization of a resource is calculated using the following equation.

$$ru_j = \frac{\sum_{i=0}^k te_i - ts_i}{T} \quad (2.5)$$

Here,  $k$  is the set of tasks assigned to resource  $j$ ,  $ru_j$  is the resource utilization rate of a resource  $j$ ,  $te_i$  is the end time of executing a task  $t_i$  on resource  $m_j$ ,  $ts_i$  is the start time of executing task and  $t_i$  on resource  $m_j$ .

$T$  is the turnaround time of an application. It is obtained using the following equation:

$$T = (\max(te_i) \forall i = 1 \dots n) - (\min(ts_i) \forall i = 1 \dots n) \quad (2.6)$$

The average resource utilization rate is defined as:

$$ru = \frac{\sum_{i=1}^m ru_i}{m} \quad (2.7)$$

Here,  $ru$  is the range between 0 to 1.

### 2.3.1.4 Load Balancing Level

The mean square deviation  $d$  of resource utilization  $ru$  is defined as:

$$d = \sqrt{\frac{\sum_{i=1}^m ru - ru_i}{m}} \quad (2.8)$$

Here,  $m$  is the number of resources. The load balancing level  $\beta$  is determined through the relative deviation of  $d$  over  $ru$  is defined as:

$$\beta = 1 - \frac{d}{ru} \times 100 \quad (2.9)$$

The most effective load balancing is achieved when  $d$  is equal to zero and  $\beta$  equals to 100%.

The other Grid model that has recently been researched is called as economic Grid. It follows the market model where user has to pay-per-use based on his/her QoS satisfaction level. QoS are defined by various parameters like execution cost, execution time, bandwidth, reliability, time within budget, and budget within the deadline, etc.

## 2.3.2 Categories of Task Scheduling Algorithms

Scheduling algorithms are classified based on when the scheduling decision is taken, what is the type of task (dependent or independent) and what are the scheduling objectives. Following section elaborates some of these categories of scheduling algorithms.

### 2.3.2.1 Dependent vs. Independent

When the relations among tasks in a Grid application are considered, a common dichotomy used is dependency vs. independency. Dependent tasks have precedence orders, that is, a task cannot start until all its parent tasks are done. Examples of dependent task scheduling algorithms are Heterogeneous Earliest-Finish-Time (HEFT) (Topcuoglu *et al.* 2002), Cluster and Duplicate based scheduling algorithm (Wieczorek *et al.* 2008), (Kang & Agrawal 2000), (Bajaj & Agrawal 2004), etc. Independent

tasks do not have relation with other tasks. (Braun *et al.* 2001) have studied various on-line and offline independent tasks scheduling heuristics i.e., Minimum Completion Time (MCT), Minimum Execution Time (MET), Min-min, and Max-min.

### 2.3.2.2 Online vs. Offline

Online mode, a task is mapped onto a machine as soon as it arrives at the scheduler. In the offline mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. A meta-task can include newly arrived tasks that arrive after the last mapping event and that were mapped in earlier mapping events, but did not begin execution. Examples of online and offline task scheduling algorithms are MCT, MET, Min-min, and Max-min respectively.

### 2.3.2.3 Meta-heuristics vs. Heuristics

Heuristics and meta-heuristics are the approximate methods used for solving non-deterministic polynomial time problems. Meta-heuristics are problem-independent techniques. Example of meta-heuristics are Evolutionary and Genetic algorithms (Zomaya & Teh 2001), Simulated Annealing (Bandyopadhyay *et al.* 2008), etc. Heuristics are problem-dependent techniques. Heuristic are designed for the solution to a specific problems. Common examples of scheduling heuristics are Min-min, Max-min, and Suf-frage (Braun *et al.* 2001). In general, meta-heuristic approaches manage to obtain much better performance, but take a longer execution time.

### 2.3.2.4 Resource Oriented vs. Application Oriented

The two major parties in Grid computing, namely, resource consumers who submit various applications, and resource providers who share their resources, usually have different motivations when they join the Grid. These incentives are presented by objective functions in scheduling. Currently, most of the objective functions in Grid computing are inherited from traditional parallel and distributed systems. Grid users are concerned with the performance of their applications, for example the makespan, flow time, and cost to

run a particular application. Examples of application oriented scheduling algorithms are Min-min, Cost optimization (Buyya *et al.* 2002), and Time optimization (Buyya *et al.* 2002). Resource providers concern is about the resource utilization, resource balance and total reward in a particular period (Thain *et al.* 2005).

### 2.3.2.5 Single Objective vs. Multi-objective

Scheduling algorithms are also classified based on the number of objectives i.e. single objective and multi-objective. Single objective algorithms minimize/maximize one objective, whereas multi-objective minimizes/maximizes more than one objectives. Single objective meta-heuristics generates one optimal solution where multi-objective meta-heuristics generates many solutions that are called Pareto front. Examples of single objective meta-heuristics are Genetic and Simulated Annealing techniques. Strength Pareto Evolutionary Algorithm (SPEA2) (Deb 2007), Pareto Archived Evolution Strategy (PEAS) (Deb 2007), Particle Swarm Optimization (PSO) (Pandey *et al.* 2010), Non-dominated Sorting Genetic Algorithm-II (NSGA-II) (Deb 2007), and Steady-State NSGA-II (SNSGA) (Durillo *et al.* 2009) are some examples of multi-objective meta-heuristic approaches.

### 2.3.2.6 Best Effort vs. QoS Constraint

Best effort based scheduling attempts to minimize the execution time of jobs, ignoring other factors such as the monetary cost of accessing resources and various users' QoS satisfaction levels. On the other hand, QoS based scheduling algorithm attempts to minimize performance under most important QoS constraints, for example, Time minimization under budget constraints (Buyya *et al.* 2002) and Cost minimization under deadline constraints (Buyya *et al.* 2002).

## 2.3.3 Online Independent Task Scheduling Algorithms

### 2.3.3.1 Random Scheduling Algorithm

Random scheduling algorithm selects a resource randomly. It does not need any information about jobs or resources. However, it usually provides poor results. In a context

where all jobs and resources are similar, it should provide an acceptable load balancing approach.

### 2.3.3.2 Round-Robin (RR) Scheduling Algorithm

Round-robin scheduling algorithm selects resources one at a time and starts over when all resources have been selected. This algorithm requires no information about jobs or resources. In a homogeneous context, RR scheduling algorithm will make a good load balancing amongst the resources. However, in a heterogeneous context, some resources will get overloaded and others will be endorsed. Indeed, each resource will get the same amount of work, but some will require more time to process their part of the work.

### 2.3.3.3 Optimistic Load Balancing (OLB) Scheduling Algorithm

OLB (Braun *et al.* 2001) scheduling algorithm assigns each task on the resource expected to be available first. The objective of this algorithm is to keep the platform as busy as possible. This algorithm only requires information about the resources state. It does not take the execution time into account. Thus, OLB scheduling algorithm can lead to poor results of resource utilization.

### 2.3.3.4 Minimum Execution Time (MET) Scheduling Algorithm

MET (Braun *et al.* 2001) scheduling algorithm assigns jobs onto the resources where the job is expected to have the smallest duration. This scheduling algorithm requires an estimation of the execution time of each job on each resource. Furthermore, if it is used in a context where tasks are characterized as consistent (a machine running a task faster will run all the tasks faster), it will assign each task to the same machine.

### 2.3.3.5 Minimum Completion Time (MCT) Scheduling Algorithm

MCT (Braun *et al.* 2001) scheduling algorithm assigns each task to the machine with shorter expected completion time to accomplish it. The completion time corresponds to the sum of the time necessary for the machine to become available (in case it is already running other tasks) plus the time that it will take in order to execute the task. This

scheduling algorithm can map more than one task per resource. The mapping complexity is  $O(m)$ , since when a task arrives, all Grid machines are examined for determining the one having the shortest expected completion time for its execution.

### 2.3.4 Offline Independent Task Scheduling Algorithms

These scheduling algorithms are designed to schedule dependent tasks in a workflow. These scheduling algorithms map the tasks on resources. Three major heuristics namely Min-min, Max-min, and Suffrage have been employed for scheduling workflow tasks in Pegasus and Askalon projects.

#### 2.3.4.1 Min-min Heuristic

Min-min heuristic has a set of unmapped meta-tasks  $M$  and a set of Grid machines  $R$ . At the first step, the algorithm computes the completion time of each task of  $M$  for every machine  $R$ . Next, the algorithm searches for the lowest completion time of each task. Then selects the task which has minimum completion time among all tasks in  $M$ , and assigns it to the machine in which this performance is expected to be obtained. The mapped task is removed from the meta-task  $M$ , and the algorithm increments the expected available time of the chosen Grid resource considering the time to run the newly mapped task. This process is repeated until there are no more tasks to schedule. As MCT scheduling algorithm, Min-min heuristic also maps more than one task per node. Being  $m$  as the number of tasks in  $M$  and  $r$ , the number of resources in  $R$ , computing the completion time of each task in all machines will take  $O(mr)$ . The loop is repeated  $m$  times, leading to a total time complexity is  $O(m^2r)$ .

#### 2.3.4.2 Max-min Heuristic

Max-min heuristic works similar to Min-min heuristic. Instead of finding the minimum completion time among all tasks in  $R$ , it finds the maximum completion time among all the tasks. Max-min heuristic attempts to minimize the penalties incurred from performing tasks with longer execution time. It is also one of the heuristics implemented in SmartNet (Lifka 1995). its time complexity is similar to Min-min heuristic.

### 2.3.4.3 Suffrage Heuristic

Suffrage heuristic (Casanova *et al.* 2000) is based on the idea that a task should be assigned to a certain resource and if it does not go to that resource, then more it will suffer. The first step finds the difference between best minimum completion time and second-best minimum completion time of each task and that is called suffrage value. Then, it selects the task that has maximum suffrage value, and assigns the task on the machine that has minimum completion time. The mapped task is removed from meta-task  $M$ . This process is repeated until all tasks are assigned from meta set  $M$ . Suffrage heuristic time complexity is  $O(m^2r)$ .

### 2.3.4.4 QoS Guided Min-min Heuristic

QoS Guided Min-min heuristic (He *et al.* 2003) divides the tasks into two categories, namely high quality and low quality based on the bandwidth requirement. Min-min heuristic is applied to assign tasks, high quality bandwidth requirement tasks take precedence over low bandwidth requirement tasks.

### 2.3.4.5 High Standard Deviation First Heuristic

High Standard Deviation First (HSDF) heuristic is proposed by (Munir *et al.* 2008). This heuristic computes standard deviation of the expected execution time of meta-tasks  $M$ . A task that has high standard deviation is assigned first to the machine based on minimum expected time to finish. This process is repeated until all tasks are assigned. Time complexity of this algorithm is higher than other offline heuristics because it computes the standard deviation.

### 2.3.4.6 Segmented Min-min Heuristic

Segmented Min-min heuristic (Wu *et al.* 2000) reduces the imbalance that occurs in Min-min heuristic. Min-min heuristic assigns the smaller task first. Thus, the smaller tasks would execute first and a few larger tasks execute later, while several machines sit idle, resulting in poor machine utilization. Segmented Min-min heuristic first orders the tasks based on average expected completion time of each task. Then, tasks are divided into

equal sized segments. The segment of larger tasks is scheduled first with the Min-min heuristic. This process is repeated until all tasks are assigned. This heuristic works well when the length of tasks varies, because it first executes the long tasks and then shorter once.

#### 2.3.4.7 Resources Aware Scheduling Algorithm (RASA)

RASA heuristic (Parsa & Entezari-Malekir 2009) is a combination of Min-min and Max-min heuristics. This algorithm first finds the completion time of each task on each resource. Then, it assigns the tasks in alternative fashion using Min-min and Max-min heuristics. RASA heuristic supports concurrency in the execution of tasks and removes the deficiency of Min-min and Max-min heuristics.

#### 2.3.4.8 Preemptive Version of Min-min Heuristic

(Khalifa *et al.* 2007) have introduced Preemptive Version of Min-min heuristic. This heuristic utilizes all the idle machines. It assigns waiting tasks to the idle machine. Waiting tasks are those tasks that are waiting for the next mapping event. Idle machines are those machines whose completion time is lesser than the makespan at particular mapping event. This heuristic assigns the task to idle machines using MET scheduling algorithm and marks them as migration enabled. At the consequent mapping event relative residual time on each machine is calculated on migration enabled tasks. Tasks are mapped according to Min-min heuristic.

#### 2.3.4.9 Modified Minimum Completion Time (MMCT) Scheduling Algorithm

(Kumar *et al.* 2009) introduced MMCT task scheduling algorithm. They categorized the jobs into three categories named as short, medium, and long based on average expected completion time of a task. Short jobs are assigned using OLB scheduling algorithm and other jobs are assigned using the MCT scheduling algorithm to take advantage of resource utilization and makespan.



#### 2.3.4.10 Min-mean Heuristic

(Kamalam & Muralibhaskaran 2010) presented Min-mean heuristic. This heuristic works in two phases. First phase assigns a task based on Min-min heuristic. Second phase computes mean completion time of all machines. Then, it finds the set of machines  $X = x_i, x_i + 1, i = 1 \dots n$  whose completion time is greater than the mean completion time. Next, tasks are moved from that machines set  $X$  to other machines whose completion time is lesser than the mean completion time, to make them nearly equal to mean completion time. This heuristic obtains lesser makespan, but takes more time to assign a task.

#### 2.3.4.11 Refinery Heuristic

(Bey *et al.* 2010) introduced a makespan refinery approach to schedule unmapped tasks. This heuristic also works in two phases. In the first phase, tasks are sorted in decreasing order based on longest minimum expected execution time. After that, tasks are assigned based on the Max-min heuristic, which is called initial task scheduling algorithm. In order to reduce the overall makespan, tasks are swapped from the maximum completion time machine to other machines in the system. This process is repeated until no more swap is possible.

### 2.3.5 Dependent Task Scheduling Algorithms

When a task comprising of a job has precedence orders, a popular model applied is the Directed Acyclic Graph (DAG), in which a node represents a task and a directed edge denotes the precedence orders between its two vertices. In some cases, weights can be added to nodes and edges to express computational costs and communication costs respectively. As Grid computing infrastructures become more mature and powerful. It supports for complicated workflow applications, which can be usually modeled by DAGs. We can find such tools like Condor DAGMan, Pegasus, and Askalon.

In general, list scheduling is a class of scheduling heuristics in which tasks are assigned with priorities and placed in a list ordered in decreasing magnitude of priority. The differences among various list heuristics mainly lie in how the priority is defined and when a task is considered ready for assignment. In the next section we discuss some

popular list heuristics.

### 2.3.5.1 HEFT Scheduling Algorithm

The HEFT scheduling algorithm works in three phases. First phase, computes average weight of each task and edge because tasks can run on multiple resources. Second phase, computes upward rank of each task. The rank value is equal to the weight of the node plus the execution time of the successors. The successor's execution time is estimated, for every edge being immediate successors of the node, adding its weight to the rank value of the successor node, and choosing the maximum of the summations. At the last phase, the tasks are sorted in decreasing order and assigned to the resources based on earlier expected time to finish. The time complexity of HEFT scheduling algorithm is  $O(e \times p)$ , where  $e$  is the number of edges and  $p$  is the number of resources. HEFT scheduling algorithm is tested on Askalon.

Fast Critical Path (FCP) scheduling algorithm is developed by (Radulescu & Van Gemund 1999). It reduces the complexity of the HEFT scheduling algorithm. Instead of sorting all the tasks of workflow at the beginning, it sorts the limited number of tasks and considers limited number of processors. The processors are ideal processors or processors that send reply. The time complexity of FCP scheduling algorithm is  $O(v \log p + e)$ , where  $p$  is the number of resources,  $v$  is the number of tasks, and  $e$  is the edge.

### 2.3.5.2 Cluster and Duplication Based Scheduling Algorithms

Cluster and Duplication Based scheduling algorithms are developed to reduce the Inter Process Communication (IPC) time. These scheduling algorithms run some tasks on more than one processor with the objective to utilize the ideal resources and reduce the makespan. Task Duplication-Based Scheduling (TDS) algorithm is introduced by (Darbha & Agrawal 2008) and works on homogenous processors. TDS algorithm computes Earliest Start Time (EST), Earliest Completion Time (ECT), Latest Allowable Start Time (LAST), Latest Allowable Completion Time (LACT), Level of Task (LT), and Favorite Predecessor (FP) of each task. Based on these values, clusters are created iteratively. In

the last step LACT and LAST is used to determine whether duplication is needed or not. For example  $j$  is *FP* of task  $i$ , if  $(LACT_j - LAST_i)$  is less than  $C_{ji}$  (communication time of task  $i$  or resource  $j$ ),  $i$  will be assigned to same processor as  $j$ , and if  $j$  has been assigned to other processor, it will be duplicated on the  $task_i$  processor. The time complexity of this scheduling algorithm is  $O(v^2)$ , where  $v$  is the number of tasks.

Task Duplication-Based Scheduling Algorithm for Network of Heterogeneous systems (TANH) is developed by (Ranaweera & Agrawal 2000). It is heterogeneous version of TDS algorithm. This scheduling algorithm works in two phases. In the first phase it makes cluster of tasks, irrespective of the number of available processors. Each task computes its favorite processor. Favorite processor is a processor that has minimum completion time. Tasks are assigned to favorite processor, if favorite processor is free, otherwise tasks are assigned to second favorite processor and so on. Second phase merges the cluster if number of available processors is less than the clusters. Time complexity of this scheduling algorithm is  $O(v^2 p \log p)$ , where  $p$  is the set of processors and  $v$  is the set of tasks.

### 2.3.5.3 Dynamic Workflow Scheduling Algorithms

Dynamic Workflow scheduling algorithms consider the dynamic nature of resources. This scheduling algorithm employed workflow partition technique. This technique partitions the workflow into sub workflows which is executed sequentially. It schedules the sub workflows based on current status of the resources. Once, the sub workflow starts execution, it schedules the next sub workflow with consideration of the current status of resources. Iterative mapping first maps the workflow based on existing List scheduling algorithms. If required, rescheduling or migration is applied. Task migration (Prodan & Fahringer 2005) moves the task from the assigned resources, if better resources are found or starting execution time of the task is elapsed. Rescheduling (Sakellariou & Zhao 2004) is employed whenever the performance of resources is diminished and does not use initial mapping.

### 2.3.6 Meta-heuristic

Many real world problems, having complex search space, are difficult to be solved by traditional optimization methods. Non-traditional optimization methods like Genetic Algorithms (GA) (Goldberg 1989), Simulated Annealing (SA) (Kirkpatrick *et al.* 1983), and Ant Colony Optimization (ACO) (Dorigo *et al.* 1996) possess several characteristics that make them more preferable than traditional optimization methods for these types of problems. The following sections discuss these methods in detail.

#### 2.3.6.1 Simulated Annealing (SA)

Simulated Annealing is a probabilistic search algorithm. The term simulated annealing derives from the process of heating and then cooling a substance slowly to finally arrive at the solid state. The search algorithm simply mimics the physical process as follows.

In the early stages of the execution, the temperature is high resulting in higher probability of accepting the solutions. Jumping occurs as a way of avoiding local minima, accepting a poorly performed solution with a higher probability. Otherwise, any solution performing better than the current solution is accepted and replaced as the best solution found so far. As the execution time elapses, the temperature decreases, thus reduces the frequency of jumping. This probabilistic nature of the system guarantees the exploration of other solution space instead of terminating whenever encountering the first local optimum.

The simulation process terminates after a number of successive executions with no improvements, and returns the best solution so far. The only drawback of Simulated Annealing is the long execution time to obtain quality solutions. Although it is possible to achieve global optimum solution, it comes at a cost of slower cooling procedure and longer iteration at each temperature level. Conversely, in shorter execution time, the algorithm compromises the solution quality.

#### 2.3.6.2 Genetic Algorithms (GA)

Genetic Algorithms (GA) are based on the mechanics of natural selection and natural genetics. They combine survival of the fittest with a structured but randomized information

exchange to form a search algorithm. The GA works with an initial population of a string of variables known as chromosomes, which hold the parameters or genes and the size of population. The chromosome can be represented using binary code or decimal system and are accordingly termed as binary-coded GA or real-coded GA. There are three operators namely selection, crossover, and mutation to generate new population from the old population and one set of the process is termed as generation. In selection operator, a set of chromosomes are selected as initial parents at the reproduction stage on the basis of their fitness, subject to the constraints specific to the problem. The fittest are given a greater chance of survival with greater probability of reproducing more off-springs. The process of mating is implemented through the crossover operator. Mutation, an arbitrary change of the genes, is implemented to preserve the genetic diversity in the population. The probability of occurrence of mutation is kept low as it can potentially disrupt a good solution.

A stochastic selection process, biased towards the fitter individuals, is implemented to select a new set of population for the next generation. Tournament selection operator is generally used for the selection of good solutions. The newly-created population is evaluated and tested for termination to decide the maximum number of generations. If the termination criterion is not met, the population is iteratively operated further by the above three operators and evaluated. This process is continued until termination criterion or a preset maximum number of generations is reached. The main feature of GA is its ability to operate on many solutions simultaneously, thereby exploiting the search space of the objective function intensively.

### 2.3.6.3 Combined Heuristics

GA can be combined with SA to create combinatorial heuristics. For example, The Genetic Simulated Annealing (GSA) (Zheng *et al.* 2006) heuristic is a combination of the GA and SA techniques. In general, GSA follows procedures similar to the GA outlined in section 2.3.6.2. However, for the selection process, GSA uses the SA cooling procedure and system temperature and a simplified SA decision process for accepting or rejecting a new chromosome.

### 2.3.7 Approaches of Meta-heuristic

(Braun *et al.* 2001) have studied heuristics and meta-heuristics on independent tasks. The fitness function of Genetic Algorithm (GA) and Simulated Annealing (SA) algorithms are taken as makespan. GA and SA are different in acceptance of off-spring. GA accepts the off-spring for next generation if it has better fitness value than existing individual in population. While SA may accept worst solution at initial stage, it depends on probability function. Their simulation results show that meta-heuristics performs better than simple heuristic, but takes more time.

(Braun *et al.* 2008) has applied Generator style genetic algorithm on dependent tasks where each task has priority, deadline, and version number. Generator style genetic algorithms is different from Steady State genetic algorithms and GA. Steady State genetic algorithms generate off-spring in each iteration. The generated off-spring occupies the place in population, if the fitness value is better than any chromosome in population. GA generates  $N$  off-springs and make a sorted list of combined population based on fitness function. Generator style generates less than  $N/2$  off-springs, merge the population with new off-springs based on fitness function. It has been observed that steady state algorithms take more time to compute and produce better results.

(Yarkhan & Dongarra 2002) have applied Simulated Annealing on Grid Application Development Software (GrADS). They have compared SA with an Adhoc Greedy scheduler. The goal of their work is to minimize execution time, regardless of any cost or time constraints. The authors concluded that SA generates schedules that have better estimated execution times than those generated by Adhoc Greedy scheduler.

### 2.3.8 Multi-objective Meta-heuristic

Many of the real world problems are generally characterized by the presence of many conflicting objectives. Therefore, it is necessary to look at that problem as a multi objective optimization problem. Pareto-optimal (non-dominated or non-inferior) solutions can be obtained for the multiple objectives optimization problems using multi objective optimization techniques (Deb 2007), (Bandyopadhyay *et al.* 2008). The solutions belonging to the Pareto-optimal solution set are not dominated by rest of the solutions in the

search space. Any solution of the Pareto optimal front cannot be said to be better than the other solutions in absence of any further information on preference ordering. Therefore, there is a demand to generate as much as Pareto-optimal solutions as possible to give more option to the planner. Classical optimization methods have limitation in handling multi-objective optimization problems. Generally, classical optimization methods convert the multi-objective optimization problem into a single objective optimization problem to generate only a single Pareto optimal solution.

### 2.3.8.1 Principle of Multi-objective Optimization (MOO)

The MOO problem consists of a number of objectives and several equality and inequality constraints (Deb *et al.* 2000) as shown follows:

$$f(x) = f_1(x), f_2(x), \dots, f_n(x) \quad i = 1, 2, 3, \dots, n \quad (2.10)$$

$$\text{Subjected to } g_i(x) \geq 0 \quad i = 1, 2, 3, \dots, m \quad (2.11)$$

$$h_i(x) = 0 \quad i = 1, 2, 3, \dots, h \quad (2.12)$$

Here  $f(x)$  is the decision variable vector representing a feasible solution, i.e. satisfying the  $m$  inequality constraints and  $h$  equality constraints;  $f_i$  is the  $i^{\text{th}}$  objective function to be minimized,  $n$  is the number of objective functions,  $g_i$  is the inequality constraints, and  $h_i$  is the equality constraint.

For unconstrained optimization problem, a solution  $x$  dominates  $y$ , if (a) the solution  $x$  is no worse than solution  $y$  in all objectives, and (b)  $x$  is strictly better than  $y$  in at least one objective. If any one of (a) and (b) is violated, the solution  $x$  does not dominate the solution  $y$ .

The unconstrained non-dominated concept is illustrated in Fig. 2.8. Here, solution '1', '2', and '3' are the non-dominated solutions. But solution '4' is dominated by solution '2' as the solution '2' is better in one objective and is equal in other objective. On the other hand, solution '6' is also dominated by solution '2'. In this case, solution '6' is not worse than solution '2' with respect to the second objective, but the solution '2' is strictly better than solution '6' with respect to the first objective. Solution '5' is dominated by solution

'2' and '4' as '2' and '4' are better than solution '5' in both the objectives.

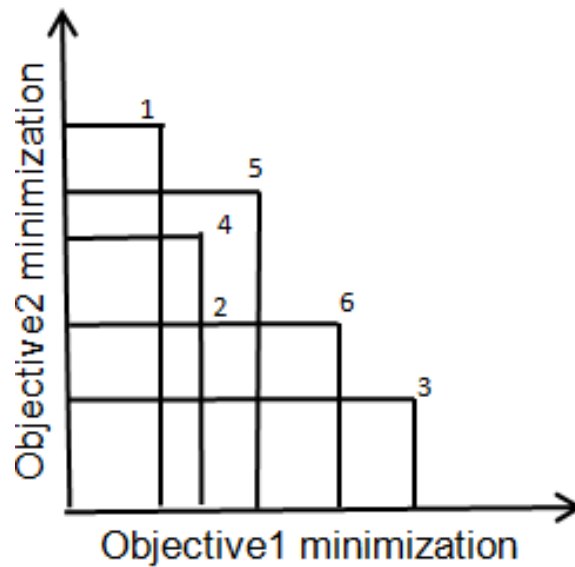


Figure 2.8: Unconstrained Non-dominated - Source: (Deb 2007)

In the recent past MOO algorithms have been developed. The objective of MOO algorithms are that the generated solutions should be diverse and convergence along the Pareto front. Thus MOO algorithm employs different strategies for diversity preservation, elitism maintains, and fitness function. Elitism can be maintained in population or external archive. Diversity is computed using fitness sharing, cell density, or crowded comparison operator (Konak *et al.* 2006). Fitness is assigned using non-domination count or based on objective functions. The following sections discuss various MOO algorithms in detail.

### 2.3.8.2 Vector Evaluated Genetic Algorithm (VEGA)

VEGA (Schaffer 1985) is the first MOO algorithm. This algorithm is similar to classical GA except that each individual objective fill the proportion of population for mating.

### 2.3.8.3 Strength Pareto Evolutionary Algorithm (SPEA)

SPEA algorithm is developed by (Zitzler & Thiele 1999). It is Evolutionary Multi-objective Optimization (EMOO) algorithm that uses varying size archive and auxiliary population. At each iteration it moves the non-dominated solution from auxiliary population



to archive. Thus, at the initial stage archive size is small, increases selection pressure. If archive is larger than predefined limit, clustering technique is used. It computes strength value of each individual in both populations. Strength value includes non-domination count of each individual divided by size of population. The individual that is with less strength value is preferable because it covers the least number of solutions.

#### 2.3.8.4 Strength Pareto Evolutionary Algorithm 2 (SPEA2)

It is an extension of SPEA (Zitzler & Thiele 1999). It uses fixed sized populations if archive size is smaller than the predefined size. Archive is filled with dominated solution. Thus, it reduces the selection pressure at initial stage. It uses fine grained non-domination that is proportion between domination and non-domination count of each individual. Its fitness value includes both Pareto rank and count. It employs a nearest neighbor technique to estimate the density of each individual, which searches the non-dominated solutions more efficiently.

#### 2.3.8.5 Pareto Archived Evolution Strategy (PAES) Algorithm

PAES algorithm uses archive and 1 + 1 evolution strategy (i.e., a single parent that generates a single off-spring). At each iteration a parent generates an off-spring, the one which is non-dominated is further mated. If both are not dominating each other, off-spring computes non-domination with respect to the archive. All the individuals of archive that are dominated by an off-spring is removed, off-spring becomes parent for the next iteration. If off-spring does not dominate any individual of archive, then parent and off-spring density is estimated with respect to the archive. If the off-spring resides in less crowded region, it is added into the archive then it becomes parent for the next iteration. This algorithm also maintains diversity which consists of a crowding procedure that divides objective space in a recursive manner. Each solution is placed in a certain Grid location based on the values of its objectives (which are used as its "coordinates" or "geographical location"). A map of such Grid is maintained, indicating the number of solutions that reside in each Grid location. Since the procedure is adaptive, no extra parameters are required (except for the number of divisions of the objective space).

### 2.3.8.6 Non-dominated Sorting Genetic Algorithm (NSGA)

NSGA (Deb 2007) classifies the population into different front. This algorithm computes the first front and then fitness sharing of each solution with the front is computed. The solution that is of less density is more preferable. This process is repeated until all individuals of a population is assigned into front. This algorithm is not very efficient because it requires sharing parameter as input, and time complexity is very high.

### 2.3.8.7 Non-dominated Sorting Genetic Algorithm-II (NSGA-II)

NSGA-II eliminates the weaknesses of NSGA. It does not require sharing parameter to select the non-dominated solution. It uses crowded comparison operator to preserve the diversity of solutions. It does not use external archive because it preserves elitism in population itself. More detail of NSGA-II algorithm is described in Chapter 3.

Various versions of NSGA-II have been developed for example Steady State NSGA-II (Durillo *et al.* 2009), Parallel NSGA-II (Nebro *et al.* 2008), etc. Steady State NSGA-II generates one off-spring and performs non-dominated sorting between off-spring and population. Hence time complexity is more. Parallel NSGA-II works on multiple processors. Multiple processors compute the objective of solutions simultaneously, thus computation time is less. Table 2.1 elaborates various MOO algorithms.

**Table 2.1:** Multi-objective Optimization Algorithms

Algorithm	Fitness	Elitism	Diversity
SPEA	Non-domination count/population size	Varying size external population	It computes the strength value
SPEA2	Non-domination count/domination count	Fixed size external population	It computes nearest neighbor technique
PEAS	Non-domination count	External population	Cell density
NSGA	Non-domination count	Maintain in population	Fitness sharing
NSGA-II	Non-domination count	Maintain in population	Crowded comparison operator

### 2.3.9 Approach of Multi-objective Meta-heuristics

(Yu 2007) have explored constraint based multi-objective genetic algorithms on dependent tasks. Scheduling algorithm finds the trade-off between two conflicting objectives execution time and cost, while meeting user's requirement. User's requirements minimize the monetary cost while meeting user's budget constraints, or minimizing the execution time while meeting user's deadline constraints. Budget and deadline value are computed using single objective genetic algorithm and HEFT scheduling algorithm respectively. After that, relax, tight, and medium constraint is given to workflow and fitness is computed.

(Garg & Singh 2011) have applied Referenced Point NSGA-II (RNSGA-II) (Deb 2007) on workflow. The major difference between NSGA-II and RNSGA-II is that NSGA-II generates solutions over entire Pareto optimal where RNSGA-II generates the solution in user specified region. Objectives of scheduling algorithm are to minimize the time and cost together with the maximization of reliability in the given QoS constraints.

(Chitra *et al.* 2011) have applied Hybrid NSGA-II on workflow where objectives are to minimize makespan and improve reliability. This algorithm uses simple neighborhood search algorithm and weighted fitness function for local search. Simplest neighborhood search algorithm starts from an initial solution and explores the vicinity of this solution using a certain mechanism to generate neighboring solutions. Neighbors are then accepted to replace the initial solution if they improve upon it. They have also used fuzzy logic to choose the best solution among all available solutions.

(Garg *et al.* 2008) have introduced Hybrid Genetic Algorithm on independent tasks, where each application requires more than one Central Processing Unit to execute. If an application is not able to find the number of required CPUs, the task is considered as infeasible and scheduled in the next iteration. This algorithm finds the initial scheduling using linear programming without considering the CPU requirement, then converts this assignment into CPU requirement, if possible, otherwise assigns the task to dummy node for the next iteration. This initial schedule is input of GA.

### 2.3.10 Utility/Quality of Service Based Scheduling Algorithms

QoS are a constraints that are related to the service. In telephony and computer networks response time, signal-to-noise ratio, and cross-talk are considered as QoS parameters. In the Grid environment deadline, price, execution time, and overhead are considered as QoS parameters. Utility is a concept, originally from economics that evaluates the satisfaction of a consumer while using a service. In a Grid environment, utility can be combined with QoS constraints in order to have a quantitative evaluation of a user's satisfaction and system performance. The classical scheduling algorithms presented in the previous section do not consider QoS or utility demands. Next, we present some QoS and utility based scheduling algorithms for Grid environments.

### 2.3.11 Approach of QoS Based Scheduling Algorithms

(Buyya *et al.* 2002) introduced Deadline and Budget Constrained (DBC) scheduling algorithms with four different optimization strategies, namely Cost optimization, Cost-time optimization, Time optimization, and Conservative-time optimization for scheduling task-farming applications on geographically distributed resources. Time optimization scheduling algorithm completes the jobs as soon as possible within the budget limit. This scheduling algorithm sorts the tasks based on the completion time. Then, it assigns a job to the first resource for which cost per job is less than or equal to the remaining budget per job. This process is repeated until all the jobs are assigned to the resource. Cost optimization scheduling algorithm completes an experiment at minimum cost within the time limit. Cost optimization scheduling algorithm sorts the resources by increasing cost. Then, for each resource in order, assign as many jobs as possible to the resource, without exceeding the deadline. The Cost-time optimization scheduling algorithm is similar to Cost optimization scheduling algorithm, but if there are multiple resources with the same cost, it applies time optimization strategy while scheduling jobs on them. The Conservative-time optimization scheduling algorithm is similar to the time-optimization scheduling strategy, but it guarantees that each unprocessed job has a minimum budget-per-job.

(Amudha & Dhivyaprabha 2011) proposed a QoS Priority-based scheduling algorithm in which QoS parameter is used as priority which is user defined. This algorithm classifies

the tasks into four groups. Groups are named as high complexity and high priority, low complexity and high priority, high complexity and low priority, and low complexity and low priority. Resources are also classified into two groups for example, high processing speed (group-1) and low processing speed (group-2) systems. High priority tasks are assigned before the low priority. High complexity tasks are assigned to high processing speed system and vice versa.

(Chen & Zhang 2009) proposed an Economic Grid resource scheduling algorithm. It is based on a utility function that includes user specified budget and deadline. This algorithm assumes Grid is hierarchical. At the top level, Grid Resource Manager (GRM) is responsible for mapping the tasks. Domain Resource Manager (DRM) is responsible for other DRM or Computing Node (CN). This scheduling process starts whenever user submits tasks to GRM. GRM gets updated information about DRM and CN, computes the utility function on different DRM, the DRM, which has maximum utility is selected for assignment. If maximum utility is not unique, it selects the DRM that has maximum variance. This process is further repeated until next level is not CN.

(Yu 2007) considered the scheduling problem on the utility Grid where user has to pay money for services. It is constraint based scheduling algorithm in which cost should be minimized, while meeting the user specified deadline for executing a workflow. This algorithm first partitions a workflow into sub-workflows. A sub-workflow contains a sequential set of tasks between two synchronization tasks (as specified in section 1.8) in the workflow. It assigns a sub-deadline to each partition by a combination of Breadth-first search and Depth-first search with critical path analysis. Then, for each partition a planning process is applied to find the optimal mapping for which the cost is the lowest and the deadline is met. The optimal search is modeled by a Markov Decision Process and is implemented using a dynamic programming algorithm. Rescheduling is also possible on the sub-workflow, when a sub-workflow misses its sub-deadline to decrease the cost.

An identical problem (Yu 2007) is considered by (Sample *et al.* 2002). The scheduler starts with bids from resource providers for the QoS. QoS includes completion time, start time, complexity, and size of the input parameters. The scheduler takes the decision based on the Pareto optimality. The scheduler assigns the tasks to the provider based on QoS agreement. If the certainty of the completion time and cost is dropped to a threshold,

which is usually caused by performance fluctuation, rescheduling will be carried out.

(Garg *et al.* 2010) introduced Min-min Cost Time Trade-off (MinCTT), Min-max Cost Time Trade-off (MaxCTT), and Suffrage Cost Time Trade-off (Suffrage CTT) algorithms for parallel tasks. Parallel task is a task that requires more than one CPU to execute. Then, it computes the utility value of each task based on average cost and average response time. After that, tasks are assigned to the resources using Min-min or Max-min or Suffrage heuristic.

(Li *et al.* 2010) proposed a cost and time balancing scheduling algorithm for parallel tasks. They have defined a new parameter urgency of the task based on deadline. Resources are divided into two groups based on average cost of schedule. Group-1 has the resources that have more cost than average cost of schedule. Group-2 has the resources that have less cost than average cost of schedule. This scheduling algorithm finds the two tasks that have minimum utility value, then checks the urgency value of those tasks. The task that has more urgency is assigned to group-2 and vice versa. Because group-2 has faster resources. This approach generates more balanced schedule.

(Braun *et al.* 2002) have developed static mapping heuristic for QoS on economic Grid. QoS parameters include timeliness, reliability, security, data accuracy, and priority. This scheduling algorithm is designed from two different perspectives, namely the user and the system perspectives with penalty. They have also classified the tasks as hard, soft and best effort tasks. Hard tasks should complete before deadline otherwise its utility value is zero. Soft tasks utility value is decreased if it does not complete within deadline. Best effort tasks utility value is always one whether it completes within deadline or not. Various utility functions have been designed for QoS parameters like task type (whether it is hard or soft) reliability and timeliness. User defines the budget and QoS parameters. If scheduler is able to find a machine that has better or equal QoS than user defined QoS tasks are scheduled on machine. Otherwise task is considered as failure.

(Golconda *et al.* 2004) have compared various static QoS heuristics on independent task. They have considered same task parameters and utility functions defined in (Braun *et al.* 2002). This scheduling algorithm first, finds the weighted utility function of a hard task, on most preferred version. Because hard task should complete within deadline and most preferred version of task has least execution time. If no machine can

**Table 2.2:** Evolution of Scheduling Algorithms

Criteria	Traditional	Cluster	Grid
Chronology	Late 1970	Late 1980	Mid 1990
Single System Image	Yes	Yes	No
Number of Nodes	Static	Static	Dynamic
Communication	Bus, Switch	LAN	WAN
Node Heterogeneity	No	Low	Yes

complete the hard task within the deadline on most preferred version, task is considered as failure. Otherwise the machine, which gives maximum utility, is assigned to the tasks. This process is repeated for soft and best effort tasks respectively. The efficiency of this approach is evaluated in terms of number of satisfied users, makespan, and utility value with the Suffrage and Min-min heuristic.

### 2.3.12 Comparison of Scheduling Algorithms

Table 2.2 shows the evolution of different types of scheduling algorithms. Traditional parallel and distributed computing scheduling algorithms provide single system image, nodes are homogenous, number of nodes are static and connected through bus and switches. Cluster scheduling algorithms also provide single system image, nodes in a cluster have low heterogeneity, communicate over LAN, and number of nodes are pre-determined. Grid scheduling algorithms do not provide single system image, nodes are dynamic and communicate over WAN.

Table 2.3 summarizes the various scheduling algorithms that are used in Grid projects.

### 2.3.13 Shortcoming of Existing Scheduling Algorithms

- Very few online heuristics consider the processor fragmentation, time, and cost of service providers simultaneously. We have developed Parallel task scheduling algorithm on economic Grid that selects the resource that leaves the smallest fragment of processor, and takes the minimum cost and time.
- Little attention has been paid on dynamic scheduling algorithms in Grid. We have designed a scheduling algorithm that model a scheduling algorithm as a state transition diagram and duplication candidate task is chosen intuitively to avoid imprac-

**Table 2.3:** Overview of Various Scheduling Algorithms

Scheduling	Algorithm	Project	Organization
Online Independent	Myopic	Condor and DAG Man	University of Wisconsin Madison, USA
Offline Independent	Min-min	GrADS Pegasus	Rice University, USA University of Southern California, USA.
List	HEFT	Askalon	University of Innsbruck, Austria.
Cluster and Duplication	TANH	Ranaweera and Agrawal	University of Cincinnati, USA
Genetic Algorithm		Askalon	University of Innsbruck, Austria.
Simulated Annealing		ICENI	London e-Science Centre, UK.

tical duplication.

- Existing Sender-initiated scheduling algorithm does not work in high system load because polling activity itself increases the system load. Thus, we proposed Enhanced Sender-initiated scheduling algorithm that uses polling information to determine the threshold. This approach improves the turnaround time of tasks and communication overhead.
- Existing improved NSGA-II algorithms either applied the pre-selection operator or memetic operator. We have combined both the operators and work on scheduling algorithm where prices offered by resources providers are not correlated with their services. Since, NSGA-II is multi-objective, it produces many solutions, it is nearly impossible to find the best solution that has minimum cost and time. Thus, we propose ranking algorithm to select the best solution.

The discussion in this chapter presents a broad picture of scheduling algorithms in the distributed environments. Together with Chapter 1, it forms the foundations of this thesis by providing the background knowledge and identifying research problems. From the next chapter, discussions will be focused on original works in the Ph.D. study.



## Chapter 3

# Dependent Task Scheduling

This chapter focuses on our proposed multi-objective dependent task scheduling algorithm on economic Grid where user has to pay-per-use. Where prices offered by resources providers are not correlated with their services. Thus, user wants more information such as range of cost and time before making decisions. For example, users may prefer solutions which have slightly longer time but offer large savings in execution cost. The proposed Double Hybrid NSGA-II (DHNSGA-II) algorithm minimizes three conflicting objectives without making them single scalar objective, using NSGA-II. DHNSGA-II does hybridization at two levels. At the first level, it uses Pre-selection operator and at the second level it uses Memetic operator/Local search. Pre-selection operator seeds the DHNSGA-II with the previously generated solutions. Memetic operator improves the current population using simple neighborhood search algorithm. Apart from DHNSGA-II we introduced an approach to rank the Pareto frontiers because Pareto frontier has many solutions; it is nearly impossible to choose the best solution. Various versions of NSGA-II is tested and results are compared.

### 3.1 Motivation

Existing seeded NSGA-II (Yu 2007) algorithm works on Utility Grid where service cost and time are reciprocal and user has to define cost and deadline. Memetic NSGA-II (Chitra *et al.* 2011) algorithm applies Simple Neighborhood Search (SNS) and minimizes the makespan and reliability. (Garg & Singh 2011) applied the reference point NSGA-

II that requires user input. (Chitra *et al.* 2011) has also introduced decision maker that suggest best weights for the different objectives using fuzzy logic. While we introduced an approach to rank the solutions based on user defined trade-off factor. Rank of solutions is computed using Technique for Order Performance by Similarity to Ideal Solution (TOPSIS) method (Sen & Yang 1998) because TOPSIS method requires few numbers of input (weights of criteria) to choose the best alternative than other Multiple Criteria Decision Making (MCDM) (Aruldoss *et al.* 2013) methods. Time complexity of TOPSIS is also lesser than other MCDM methods because it does not perform pairwise comparison of each criterion with other criteria. Following sections illustrate non-dominated sorting and crowded comparison operator.

### 3.1.1 Steps of NSGA-II

NSGA-II has the following steps:

1. Initialize a population using uniform random distribution method.
2. Apply crossover and mutation operators to generate children solutions.
3. Combine the children and parent population to compute non-dominated sorting.
4. Compute the objective value of each solution.
5. Compute the non-domination rank of each solution and assign different fronts. The solutions having lesser rank are better candidates for next generation.
6. Compute the crowding distance of each solution within the front. For a minimization type optimization problem, a solution  $x$  wins with another solution  $y$  if (a) solution  $x$  has better rank than solution  $y$ , or, (b) if the solutions  $x$ , and  $y$  have the same rank, but solution  $x$  has large crowding distance than solution  $y$ .

### 3.1.2 Crowded Comparison

Crowded comparison operator uses non-domination count of each solution in a population. Then, it makes different fronts of solutions based on non-domination count. Each solution in a particular front computes the density of solutions with other solutions in

the front. Density of an individual is computed using average distance of two points on either side of this point along each objective of the problem. This value is called crowding distance. After that, the solution that resides in the less crowded region is preferred in a particular group.

### 3.2 Problem Definition

This section formulates the Grid resource scheduling problem into Grid resource market model. It considers the economic Grid that is a collection of heterogeneous clusters and network resources. Clusters are heterogeneous in processor architecture and pricing. Network resources have different speed and cost. We use Directed Acyclic Graph (DAG) to model an application as shown in Fig. 3.1a. A workflow  $w$  is represented by a DAG ' $G$ '  $= (v, e, x, c)$ , where  $v$  and  $e$  are the set of tasks and directed edges respectively. A node in the task graph represents a task that runs non-preemptively on any cluster. Each edge is denoted by  $e_{ij}$  corresponding to the data communication between  $t_i$  and  $t_j$ , where  $t_i$  is called immediate parent task of  $t_j$ . Child task cannot be started until all of its parent tasks are completed. A task which does not have a parent task is called entry task  $t_{entry}$ . A task that does not have a child task called exit task  $t_{exit}$ .  $x$  is computation matrix in which  $x_{ij}$  is computation time of task  $i$  on cluster  $j$ .  $c$  is the communication matrix shown where  $c_{ij}$  is the communication time between  $(t_i, t_j)$ . Fig. 3.1a, shows example of an application modeled by DAG ' $G$ '  $= (v, e)$  where  $v = (t_1, t_2, \dots, t_n)$  are set of eight tasks to be executed. Fig. 3.1b represents computation matrix  $x$ , of three clusters named as  $M_1, M_2$  and  $M_3$ . Fig. 3.1c represents  $c$  communication requirement of between sub tasks. A schedule is a function  $s : v \rightarrow m$  that maps  $v$  tasks on  $m$  clusters, that executes it.

Completion time of task: The completion time  $com_{ij}$  of a task  $t_i$  on the cluster  $c_j$  is given by

$$com_{ij} = st_{ij} + x_{ij} \quad (3.1)$$

Here,  $st_{ij}$  is the start time of the task  $t_i$  on cluster  $c_j$  and computation time  $x_{ij}$  is added. Start time of the entry task is zero. Other tasks start time is computed by considering the completion time of all immediate predecessors of the task. The communication time  $c_{ij}$ ,

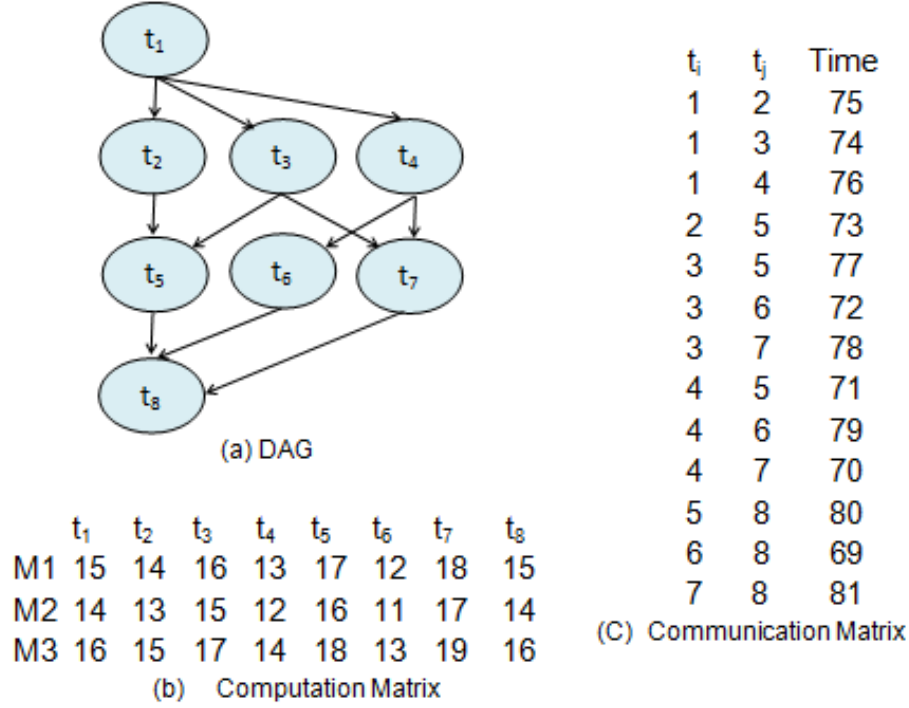


Figure 3.1: Workflow of 8 Tasks

is added if the dependent tasks are allocated to different clusters.

Completion time of cluster: The completion time  $com_j$  of cluster  $c_j$  is the maximum of completion time  $com_{ij}$  of all assigned tasks is given as

$$com_j = \max (com_{ij}) \quad \forall i = 1 \dots k \quad (3.2)$$

Here  $k$  is the set of all assigned tasks to the cluster  $c_j$ .

Makespan: Makespan of the workflow is the maximum of  $com_j$ . It is defined as follows

$$makespan (sch) = \max (com_j) \quad \forall j = 1 \dots m \quad (3.3)$$

Here,  $com_j$  is the completion time of cluster  $c_j$ ,  $m$  is the number of clusters, and  $sch$  is schedule of the workflow.

Computation Cost: It occurs when a task is executed on a cluster. The computation cost of task  $t_i$  on cluster  $c_j$  is determined by

$$compuCost_{ij} = compuCost_j \times x_{ij} \quad (3.4)$$

Here,  $compuCost_j$  is cluster  $c_j$  computation cost in G\$ (G\$ means Grid Dollar), and  $x_{ij}$  is computation time of task  $i$  on cluster  $j$ . Computation cost  $compuCost$  of schedule  $sch$  is calculated as follows:

$$compuCost(sch) = \sum_{i=1}^n compuCost_{ij} \quad (3.5)$$

Here,  $n$  is the number of tasks.

Communication Cost: It occurs when a cluster transfers result of a task to other clusters. The communication cost is defined as:

$$commuCost_{ij} = commuCost_j \times c_{ij} \quad (3.6)$$

Here,  $commuCost_j$  is cluster  $c_j$  communication cost in G\$, and  $c_{ij}$  is the communication time between task  $i$  and  $j$ . Communication cost is not reciprocal of communication time. Communication cost  $commuCost(sch)$  of schedule  $sch$  is computed as follows:

$$commuCost(sch) = \sum_{i=1}^n commuCost_{ij} \quad (3.7)$$

The task scheduling problem is formulated by considering the objectives of minimizing the makespan, computation cost and communication cost of the schedules.

Given an application graph  $G$  and a set of clusters  $m$ , the task scheduling problem is to determine a static distributed schedule with a minimal makespan, computation cost and communication cost simultaneously. This is a multi-objective scheduling problem. In multi-objective optimization, optimality is not defined as an absolute best solution. The notion of pareto dominance is considered in order to get a partial order between solutions. The quality of a solution is represented by the values of all the objective functions. This problem is formulated as a non-linear multi-objective optimization problem as follows:

$$Min f = [f_1, f_2, f_3] \quad (3.8)$$

### 3.2.1 Objective functions

1. Makespan objective: The makespan of a schedule  $sch$  is calculated as:

$$f_1 = makespan(sch) \quad (3.9)$$

where  $makespan(sch)$  is calculated using equation (3.3). Makespan is the completion time of a schedule  $sch$ .

2. Computation cost: The computation cost of a schedule  $sch$  is calculated as:

$$f_2 = compuCost(sch) \quad (3.10)$$

where  $compuCost(sch)$  is calculated using equation (3.5). Computation cost is the cost of computation resources to execute a schedule  $sch$ . Computation cost is measured in G\$.

3. Communication Cost: The communication cost of a schedule  $sch$  is calculated as:

$$f_3 = commuCost(sch) \quad (3.11)$$

where  $com(sch)$  is calculated using equation (3.7). Communication cost is the cost of network resources to transfer data from one cluster to other cluster. Communication cost is measured in G\$.

## 3.3 Double Hybrid NSGA-II (DHNSGA-II)

NSGA-II is similar to GA. NSGA-II works on multi-objective while GA works on single objective and provides single best solution at the end. Pseudo code of GA is given in Algorithm-2. Initially it generates random solutions. Then, it performs crossover and mutation operators in order to generate offspring. This process is repeated until  $i$  is less than population  $popSize$ . Then, it computes objective function of each offspring. This process is repeated until  $gen$  is less than generation  $N$ .

DHNSGA-II is an enhancement of NSGA-II where solutions are seeded and memetic

operator is applied on population. Pictorial diagram and pseudo code of DHNSGA-II are shown in Fig. 3.2 and Algorithm-3 respectively. Its, initial population  $p'$  is seeded through GA solutions (lines 3-5). DHNSGA-II minimizes three objectives namely communication cost, computation cost, and makespan. Thus, GA runs three times to minimize each objective. After that remaining solutions are generated randomly. In order to compute offspring  $p''$ , parents  $p$  are selected from population  $p'$  using Binary Tournament operator (line 11). Crossover and mutation operators are applied on parents  $p$  (lines 12 to 13). After that newly generated offspring  $p$  are evaluated to determine their fitness (line 14) and added into new offspring  $p''$  (line 15). This process is repeated until the offspring  $p''$  size is less than population size  $popSize$ . Next, SNS algorithm is applied to newly generated offspring (line 17) for memetic operator. Pseudo code of SNS algorithm is given in Algorithm-4.

---

**Algorithm 1** Pseudo Code of GA

---

```

1: Input =  $G(v, e, x, c)$ 
2: Output = single schedules
3:  $p' \leftarrow \text{GernatesRandomSolutions}()$ ;
4: for  $gen = 1 \rightarrow N$  do
5:   for  $i = 1 \rightarrow popSize$  do
6:      $p \leftarrow \text{selectTwoParents}(p')$ ;
7:      $p \leftarrow \text{performCrossover}(p)$ ;
8:      $p \leftarrow \text{performMutation}(p)$ ;
9:      $\text{computeObjective}(p)$ 
10:     $p'' \leftarrow \text{addOffspring}(p)$ ;
11:   end for
12: end for

```

---

SNS algorithm selects current solution  $c$  from offspring and searches its neighborhood solution  $c'$  using swap based mutation. After that, it computes the fitness function using equation 3.12 of  $c'$ . If fitness function of  $c'$  dominates, it replace the  $c$ . Fitness function of SNS algorithm is follows:

$$f_c = \sum_{i=1}^m w_i \frac{f_{in} - \min_i}{\max_i - \min_i} \quad (3.12)$$

Here,  $w_i$  is weight of  $i^{th}$  objective,  $\min_i$  is minimum value of  $i^{th}$  objective,  $\max_i$  is maximum value of  $i^{th}$  objective, and  $f_{in}$  is the fitness value of  $i^{th}$  objective of current solution. Local search operator is performed on set solutions [10, 10, 5], at set of time [1, 2, 5].

After that Non-dominated sorting between offspring and population is performed to

assign different fronts (line 18). This process is repeated until generation  $gen$  is less than number of generation  $N$ .

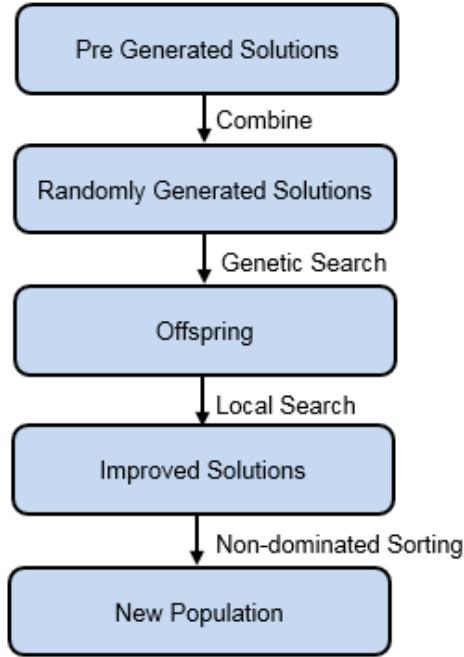


Figure 3.2: Pictorial Diagram DHNSGA-II

### 3.3.1 Implementation of DHNSGA-II

The chromosome is represented using two strings, namely matching string and scheduling string. Scheduling string represents the schedule. Matching string represents the task order. Following genetic operators are used:

#### 3.3.1.1 Chromosome Representation

The process of representing a solution that conveys the required meaning is necessary. We have used scheduling string  $Ss$ , and matching string  $Ms$  of length  $v$ . Matching string contains the value between 0 to max cluster.  $Ms[i] = k$  means the task  $v_i$  is assigned to cluster  $k$ . Scheduling String  $Ss$  is generated using topology sort of length  $v$ . Scheduling string  $Ss[i] = k$  indicates that  $v_k$  is  $i^{th}$  sub task of the DAG. For the Fig. 3.1b matching string  $Ms$  and scheduling string  $Sc$  could be  $[M1, M2, M3, M2, M3, M2, M1, M3]$  and  $[t_1, t_2, t_4, t_3, t_6, t_7, t_5, t_8]$  respectively.



**Algorithm 2** Pseudo Code of DHNSGA-II

---

```

1: Input =  $G(v, e, x, c)$ 
2: Output = multiple schedules
3: for  $i = 1 \rightarrow 3$  do
4:    $p' \leftarrow \text{seedsSolutions}()$ ;
5: end for
6: for  $i = 3 \rightarrow \text{popSize}$  do
7:    $p' \leftarrow \text{RandomlyGernatedSolutions}()$ ;
8: end for
9: for  $gen = 1 \rightarrow N$  do
10:  for  $i = 1 \rightarrow \text{popSize}$  do
11:     $p \leftarrow \text{selectTwoParents}(p')$ ;
12:     $p \leftarrow \text{performCrossover}(p)$ ;
13:     $p \leftarrow \text{performMutation}(p)$ ;
14:     $\text{computeObjectives}(p)$ 
15:     $p'' \leftarrow \text{addOffspring}(p)$ ;
16:  end for
17:   $p'' \leftarrow \text{LocalSearch}(p'')$ ;
18:   $p' \leftarrow \text{nonDominatedSorting}(p', p'')$ ;
19: end for

```

---

**Algorithm 3** Pseudo Code of SNS Algorithm (Local Search)

---

```

1: Input =  $(p'')$ ;
2: Output =  $(p'')$ ;
3:  $p \in [10, 10, 5]$ 
4:  $l \in [1, 2, 5]$ 
5: for  $count = 1 \rightarrow p$  do
6:    $c \leftarrow \text{SelectsSolution}(p'')$ ;
7:    $c' \leftarrow c$ 
8:    $f \leftarrow \text{Compute fitness of } c'$  3.12
9:   for  $k = 1 \rightarrow l$  do
10:    Apply swap mutation on  $c'$ 
11:     $f' \leftarrow \text{Compute fitness of } c'$  3.12
12:    if  $f' \leq f$  then
13:       $c \leftarrow c'$ 
14:    end if
15:  end for
16: end for

```

---

### **3.3.1.2 Selection/Replacement**

Selection phase is used to allocate reproductive trials to chromosomes according to their fitness. There are different approaches that can be applied during the selection phase. Binary Tournament selection operator is used due to its efficiency, generation of diverse population and simple implementation (Blickle & Thiele 1996). It selects the population based on the rank and crowding distance (Deb 2007), (Bandyopadhyay *et al.* 2008). An individual selected has either its rank lesser (better) than the other or its crowding distance greater than the other.

### **3.3.1.3 Crossover**

Crossovers are used to create new solutions by rearranging parts of the existing solutions in the current population. The idea behind the crossover is that the fittest solution may result from the combination of two of the current fittest solutions. Two point crossover is implemented for the matching string  $M_s$  and illustrated in Fig. 3.3. It is implemented as follows:

1. Two parents are chosen at random in the current population.
2. Two random points are selected from the matching string  $M_s$  to form a crossover window.
3. All machines included in the crossover window are chosen as successive crossover points.
4. The machines allocations of all tasks within the crossover window are exchanged.

### **3.3.1.4 Mutation**

Mutations operator is applied to obtain features that are not possessed by either of its parents. This process helps the algorithm to explore new and possibly better genetic material than previously considered. Move mutation is developed for the

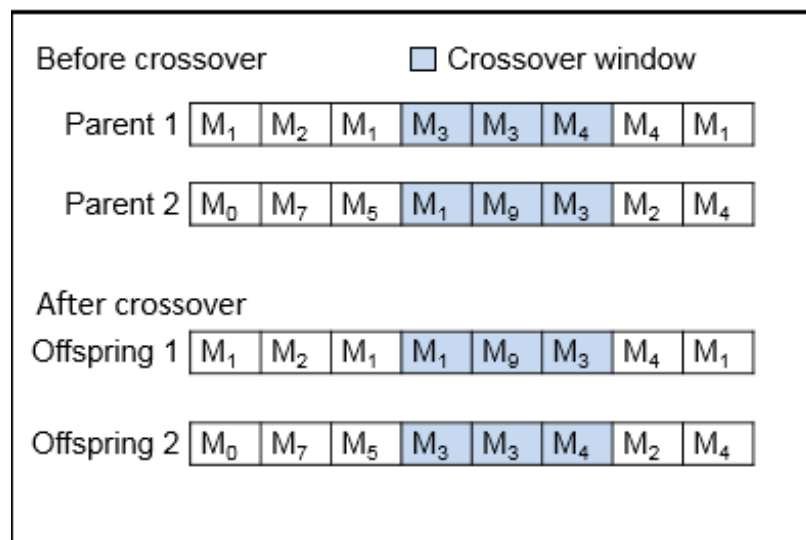


Figure 3.3: Two Point Crossover

workflow scheduling problem. The mutation operators are applied to the chosen solutions with probabilities 0.3. Two random numbers are generated in which first indicates the task number and second indicates the machine number in  $M_s$ . For example, if matching string  $M_s$  is [M1, M2, M3, M2, M3, M2, M1, M3] and two random numbers are 2 and 4. After applying the move operator new  $M_s$  will be [M1, M4, M3, M2, M3, M2, M1, M3].

### 3.3.1.5 Pre-selection

In NSGA-II, the initial population is usually generated randomly. Besides the random method, we have used GA because GA provides better solutions than other list heuristics (Aggarwal *et al.* 2005). GA pseudo code is given in Algorithm-2. Fitness function of GAs are makespan, computation cost, and communication cost as given in equations 3.9 to 3.11.

### 3.3.1.6 Local Search/Memetic Operator

Local search operator (Ishibuchi & Murata 1998) uses swap based mutation to increase the fitness measure. It randomly generates two numbers for  $M_s$  and swapped. For example, if  $M_s$  is [M1, M2, M3, M2, M3, M2, M1, M3] and two random numbers are generated like 3 and 1 that indicates the index of  $M_s$ . After

applying the swap new  $M_s$  will be [M3, M2, M1, M2, M3, M2, M1, M3].

### 3.3.1.7 Evaluation

Evaluation of population is performed over three objectives that are described in section 3.2. Non-domination count of solutions are computed in order to find different fronts. If particular front size is more than the remaining population size, it performs crowded comparisons operator for clustering and selects the remaining chromosomes.

### 3.3.2 Time Complexity

Time complexity of DHNSGA-II is similar to NSGA-II. NSGA-II time complexity of one iteration is governed by non dominated sorting that is  $O(mN^2)$ , where  $N$  is the population size and  $m$  is the number of objectives (Deb 2007). Similarly, GA one iteration time complexity is dominated by compute objective function of GA. It computes the objective value of each solution in population, therefore,  $O(N)$ . We run GA, for each objective, so it becomes  $O(k \times N)$ , where  $k$  is the number of objectives. Overall time complexity of DHNSGA-II is  $O(mN^2)$  which is dominated by NSGA-II.

## 3.4 Simulation and Evaluation

A Java based simulator has been designed using jMetal (Durillo & Nebro 2011) tool kit. jMetal stands for Meta-heuristic Algorithms in Java. jMetal toolkit provides an environment to solve multi-objective optimization problems. To test the effectiveness of the DHNSGA-II, real world DAG of Gauss Elimination algorithm is used as a workflow. Gauss elimination graph is introduced by (Topcuouglu *et al.* 2002). Gauss Elimination algorithm finds the upper triangle of a square matrix. This application requires matrix of size  $m$  as an input, that should be  $2^m$ . The total number of tasks in a Gauss elimination graph is equal to  $(m^2 + m - 2)/2$ . DHNSGA-II requires two kinds of parameters; to generate a

workflow and perform the DHNSGA-II. The parameters to generate a workflow like matrix size, average computation cost, and computation to communication ratio, etc. is shown in Table 3.1. DHNSGA-II parameters are shown in Table 3.2. It requires probability of crossover and mutation operators, etc.

**Table 3.1:** Workflow Parameters

Parameter Name	Range
Matrix size	8 to 64
Average computation cost	10 to 200
Computation to communication ratio	0.1 to 10
Heterogeneity factor of resources	0.2 to 0.5
Cost of computation resources in G\$	0.1 to 0.9
Cost of network resources in G\$	1 to 10

To assess the search capability of the proposed algorithm, we have generated 16 Gauss elimination graphs of a particular matrix. After that a particular graph, is ten times generated, each time different computation cost and computation to communication ratio is randomly chosen from particular set. Thus, a particular matrix graph is evaluated 160 times. We have compared the seeded NSGA-II, Memetic NSGA-II, and DHNSGA-II with reference solutions. Memetic NSGA-II applies the local search on the best solution. The best solutions are picked out using Roulette wheel selection method because (Noraini & G. 2011) has suggested that whenever solution quality is the main concern, then rank-based selection (Roulette wheel come under this category) strategy is the best.

**Table 3.2:** DHNSGA-II Parameters

Parameter Name	Value
Population size	100
Cross over rate	0.8
Mutation	0.3
Generation	500
Crossover operator	Two point crossover
Mutation operator	Move based mutation
Selection operator	Binary tournament
Local selection operator	Roulette wheel
Local operator	Swap

### 3.4.1 Performance Index

Multi-objective Optimization (MOO) algorithms measure two parameters regarding the obtained solution set and reference solution set. It should converge close to the reference solution set, and it should maintain diverse solution set. The first condition clearly ensures that the obtained solutions are near optimal, and the second condition ensures that wide ranges of trade-off solutions are obtained. Hyper Volume (HV) indicator (Zitzler & Thiele 1999) is used to compute both convergence and diversity.

It computes difference between non-dominated solution set obtained from algorithm and reference solution set. Reference solution set is obtained by merging all of the non-dominated solutions generated by all of the algorithms. The higher value is better for HV. Statistical significance with alpha value (0.05) is computed (Garg *et al.* 2010), (Yu 2007).

### 3.4.2 Results of DHNSGA-II

Fig. 3.4 to 3.6 show the comparison between reference solution set and non-dominated solution set obtained of matrix size 64. In the figures, communication cost (blue color), and computation cost (green color) show the reference solution set, and communication cost (red color) and computation cost (yellow color) show the solution set of different algorithms. Reference solutions have least communication cost, computation cost, and makespan. From the figures it is clearly visible that DHNSGA-II Fig. 3.6 has least computation cost, communication cost and makespan. DHNSGA-II has more number of solutions around the reference point while other solutions are scattered. This is because its initial population is seeded and local search is applied. DHNSGA-II has at least 20% less makespan, communication cost, and computation cost than other algorithms. From the figures it can be also observed that Seeded NSGA-II performs better than Memetic NSGA-II. Thus, we conclude that the pre-selection operator plays a greater role than the local search. Seeded NSGA-II has 15% less objective value than Memetic NSGA-II.

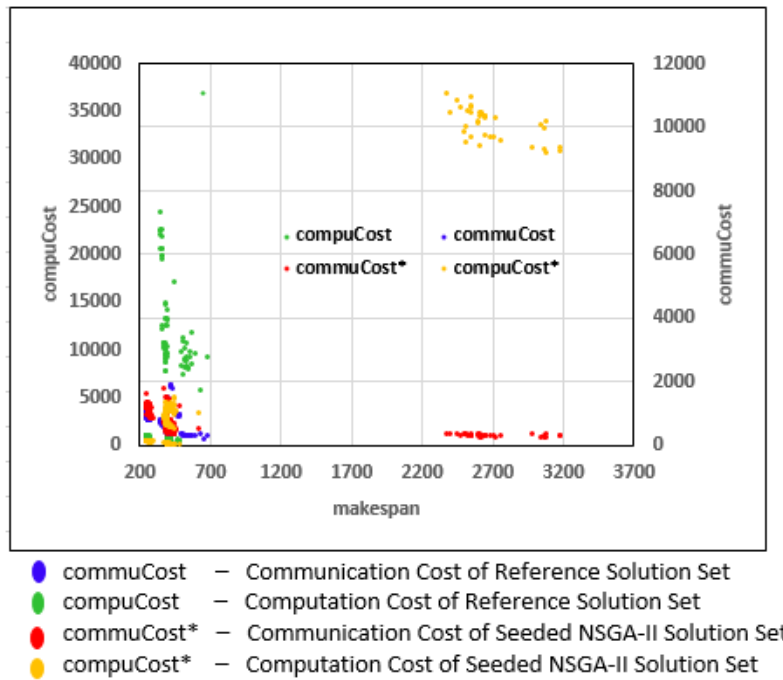


Figure 3.4: Comparison of Seeded NSGA-II Solutions with Reference Solutions

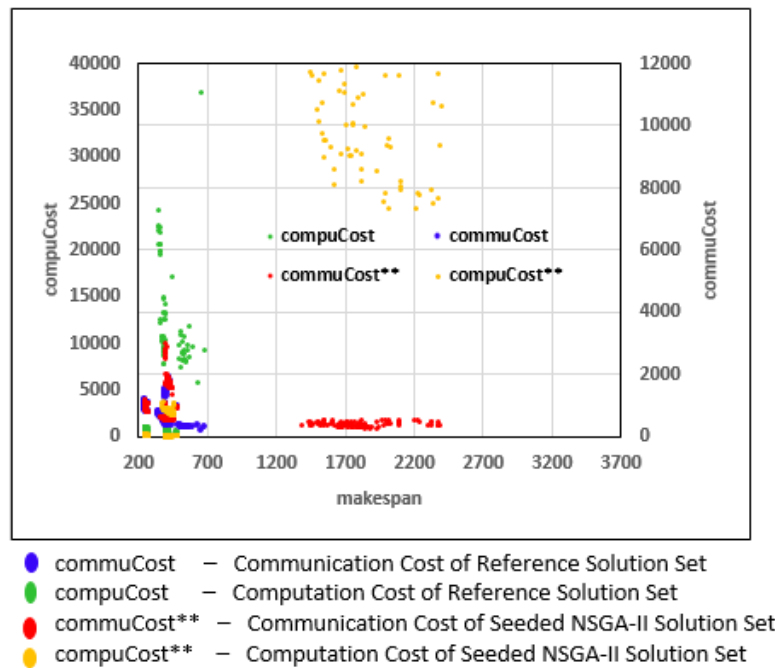


Figure 3.5: Comparison of Memetic NSGA-II Solutions with Reference Solutions

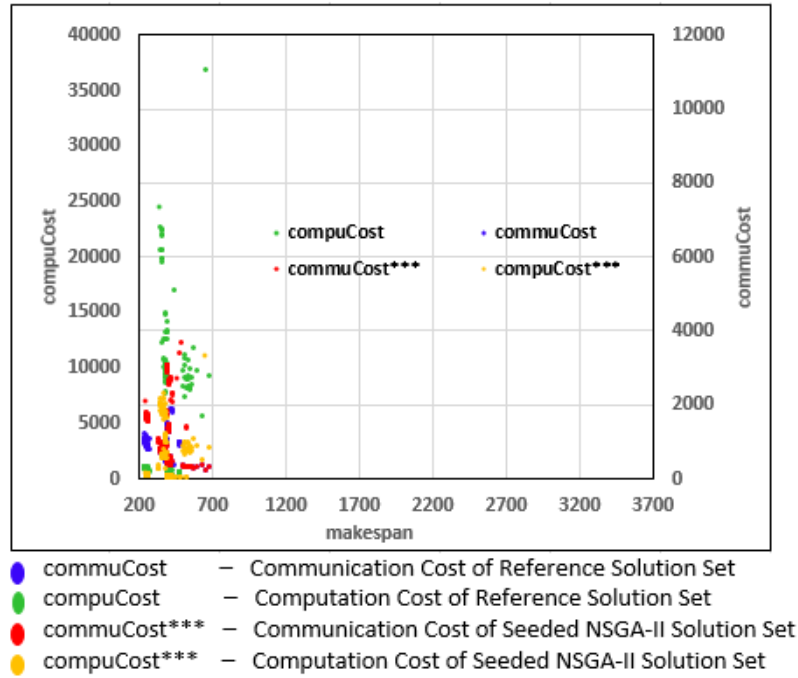


Figure 3.6: Comparison of DHNSGA-II Solutions with Reference Solutions

Inter Quartile Range (IQR) value (It measures a variability of data by ignoring outliers of different quartiles) of HV of different size of matrices is summarized in Table 3.3. Since the lesser value of IQR is better, from the table it can be observed that DHNSGA-II obtains least value than other algorithms. Seeded NSGA-II performs better than Memetic NSGA-II. DHNSGA-II overall average (all matrices) HV value is 0.11 lesser than Seeded NSGA-II. Similarly, Seeded NSGA-II overall average (all matrices) value 0.06 is lesser than Memetic NSGA-II.

Table 3.3: HV of Different Matrices

Matrix	Seeded NSGA-II	Memetic NSGA-II	DHNSGA-II
8	0.46	0.47	0.32
16	0.32	0.34	0.22
32	0.38	0.40	0.21
64	0.25	0.42	0.23

### 3.4.3 Ranking of Non-dominated Solutions

A large number of non-dominated solutions are provided by the DHNSGA-II, so the subjective ranking of solutions is very difficult and also will not be precise.



In the present investigation, a comprehensive approach has been adopted to rank the non-dominated solutions. Non-dominated solutions are ranked using TOPSIS algorithm. It gives the rank of a solution based on relative closeness with positive (maximum) and negative (minimum) of a separation matrix. Separation matrix is an M-dimensional Euclidean distance of each alternative from the positive criteria value and negative criteria value which is computed as follows:

$$R_j^+ = \sqrt{\sum_{i=1}^n (x_{ij} - x_j^+)^2} \quad j = 1 \dots m \quad (3.13)$$

$$R_j^- = \sqrt{\sum_{i=1}^n (x_{ij} - x_j^-)^2} \quad j = 1 \dots m \quad (3.14)$$

Here,  $m$  is the number of objectives,  $n$  is the number of solutions,  $x_{ij}$  is normalized value of each solution's objective,  $x_i^+$  and  $x_i^-$  is maximum and minimum value of each criteria,  $R_i^+$  is the distance from positive solution, and  $R_i^-$  is the distance from negative solution.

The relative closeness of each solution  $i$  with ideal solution  $R_i^-$  and  $R_i^+$  is measured as shown in equation (3.15).

$$C_i^+ = \frac{R_i^-}{R_i^- + R_i^+} = 0 \leq c_i^+ \leq 1; i = 1 \dots n. \quad (3.15)$$

The solution that has the least value of  $C_i^+$  is considered as the best alternative.

#### 3.4.4 Results of Ranking Algorithm

The developed approach has been tested on different matrices. Top five solutions of each algorithm are selected by giving equal weight (0.33, 0.33, 0.34). After that ratio of makespan and total cost (communication cost plus computation cost) is computed. Matrix of size 32 and 64, top five solutions, makespan, and total cost ratio of each algorithm is tabulated in Table 3.4 and 3.5 respectively. It can be observed that in the case of matrix size 32, Table 3.4 ratio of average makespan and total cost is 11.59, 13.16, and 4.18 of Seeded NSGA-II, Memetic NSGA-II, and

DHNSGA-II respectively. In the case of matrix size 64 Table 3.5 ratio of average makespan and total cost is 18.07, 22.07, and 8.08 of Seeded NSGA-II, Memetic NSGA-II, and DHNSGA-II respectively. Thus, it can be observed that Seeded NSGA-II has less objective values than Memetic NSGA-II. DHNSGA-II has 30% to 32% lesser ratio of average makespan and total cost than other techniques. DHNSGA-II outperforms the other two approaches and obtains good reduction in average ratio of makespan and total cost.

**Table 3.4:** Ratio of Makespan and Total Cost of Matrix Size 32

No.	Seeded NSGA-II	Memetic NSGA-II	DHNSGA-II
1	11.24	13.96	3.82
2	12.31	13.80	4.23
3	11.22	12.80	4.35
4	11.71	13.58	4.38
5	11.47	11.67	4.14
Average	11.59	13.16	4.18

**Table 3.5:** Ratio of Makespan and Total Cost of Matrix Size 64

No.	Seeded NSGA-II	Memetic NSGA-II	DHNSGA-II
1	17.90	23.77	8.14
2	17.81	20.64	8.17
3	17.85	21.88	7.99
4	18.09	20.87	7.90
5	18.68	23.19	8.20
Average	18.07	22.07	8.08

### 3.5 Discussion

In this chapter, workflow scheduling problem is analyzed and solutions are performed for economic Grid. Our proposed algorithm DHNSGA-II minimizes the computation cost, makespan, and communication cost. The multi-objective problem of workflow scheduling is solved using GA, NSGA-II, and neighborhood search approaches. We have compared our proposed algorithm DHNSGA-II with Seeded NSGA-II and Memetic NSGA-II. The reference solution set is obtained by combining the Pareto front of all the algorithms. Then, spread and convergence

of algorithms are measured along with reference solution set. From the results, it is noted that DHNSGA-II gives the satisfactory performance. The DHNSGA-II is further analyzed using TOPSIS to rank the solutions built upon their distance from the best solution and worst solution. Considering the ranking of the solutions, the decision manager may choose a suitable candidate among the top-ranking solutions to justify the objectives defined by the management along with the present market scenario. In the current study, neighborhood search is used as local search. In future, we plan to apply the other local search techniques like archived multi-objective based simulated annealing, etc.

## Chapter 4

# Independent Task Scheduling

This chapter purposes two novel scheduling algorithms for handling independent, serial, and parallel tasks. An independent task does not have parent and child relationship with other tasks, thus it can execute in any order. An independent task can be a parallel or a serial task. A parallel task requires more than one processor to execute the task. Proposed Enhanced Refinery heuristic works on computational Grids that minimize makespan, while Parallel task scheduling algorithm works on Economic Grids that minimize makespan, cost, and processor fragmentation.

### 4.1 Enhanced Refinery Heuristic

Enhanced Refinery (ER) heuristic is a type of offline heuristic that schedules tasks at predefined scheduled intervals.

#### 4.1.1 Motivation

Existing Min-min heuristic (Braun *et al.* 2001) attempts to minimize makespan by scheduling the smallest task on the fastest machine. Outcome of this approach, is shorter makespan, if the execution time of the tasks varies slightly. However, if there are large and small tasks, the large ones may be assigned to slower machines and the makespan of the system will be increased dramatically, which is of course

not a good heuristic.

The Max-min heuristic (Braun *et al.* 2001) seems to be better than the Min-min heuristic when the number of short tasks is much more than the long ones. For example, if there is only one long task, the Max-min heuristic executes many short tasks concurrently with the long task. In this case, the makespan of the system is most likely determined by the execution time of the long task. However, when there is more than one long task Max-min heuristic assigns the shorter task to fastest machine and longer task assigns to slower machines. The result of this approach increases the makespan.

RASA heuristic (Parsa & Entezari-Malekir 2009) is designed to eliminate deficiency of Max-min and Min-min heuristics. It applies Min-min and Max-min heuristic alternatively. It supports concurrency in the execution of tasks.

Refinery heuristic (Bey *et al.* 2010) works in two phases in order to optimally assign the tasks to machines in the Grid system. In the first phase, it arranges the tasks in the longest minimal execution time, and then assigns the tasks to the machine that takes minimum completion time. In the second phase, tasks are swapped from highest completion time machine to the other available machines in the system. The pseudo code of Refinery heuristic is given in Algorithm-5 and described as follows:

Refinery heuristic first finds the order of tasks using the latest completion time of each task (lines 2 to 4). Then it assigns the task in that order to a machine that has minimum completion time (lines 5 to 9). This process is repeated until all the tasks are assigned. Second phase computes the makespan (line 13). Next, it finds the machine that has highest completion time, that machine is named as makespan machine ( $M_m$ ) (line 15). Now, all the tasks that are assigned on  $M_m$  machine are swapped to other machines in the system, with the objective that completion time of  $M_m$  machine and other machine should be less than the makespan. This process is repeated until makespan gets reduced.

Since, Max-min and Min-min heuristics have their own drawbacks. Refinery heuristic first sorts the task, then assign using Max-min heuristic, after that swap

---

**Algorithm 4** Pseudo Code of Refinery Heuristic

---

```

1: Input (ETC[task][machine]);
2: for  $T_i = 0 \rightarrow N$  do
3:    $L[i] \leftarrow \text{computeLatestFinishtime}(ETC)$ ;
4: end for
5: for  $T_i = 0 \rightarrow N$  do
6:    $k \leftarrow \text{findMax}(L)$ ;
7:    $m \leftarrow \text{findMachine}(k)$ ;
8:    $\text{assignTask}(k,m)$ ;
9:    $\text{remove}(k,L)$ ;
10:   $\text{updateMachine}(m)$ ;
11: end for
12: repeat
13:   $\text{oldMake} \leftarrow \text{computeMakespan}()$ ;
14:   $\text{tmp} \leftarrow \text{newMake}' \leftarrow \text{oldMake}$ ;
15:   $M_m \leftarrow \text{findMachine}(\text{oldMake})$ ;
16:  for all  $T_i \in M_m$  do
17:    for all  $M_j \notin M_m$  do
18:      for  $T_k \in M_j$  do
19:         $\text{newMake}'' \leftarrow \text{computeFinishTime}(M_j)$ ;
20:         $\text{newMake}'' \leftarrow \text{newMake}'' - \text{ETC}[T_k][M_j] + \text{ETC}[T_i][M_j]$ ;
21:         $\text{newMake}' \leftarrow \text{newMake}' + \text{ETC}[T_k][M_m] - \text{ETC}[T_i][M_m]$ ;
22:        if  $\text{newMake}'' \leq \text{oldMake}$  then
23:          if  $\text{newMake}' < \text{oldMake}$  then
24:             $\text{swapTask}(T_i, T_j)$ ;
25:             $\text{oldMake} \leftarrow \text{newMake}'$ ;
26:          end if
27:        end if
28:      end for
29:    end for
30:  end for
31: until ( $\text{oldMake} < \text{tmp}$ );

```

---

the task from one machine to other machines. In Refinery approach, number of tasks at each machine do not change because it swaps the tasks between two machines. While we propose an Enhanced Refinery heuristic that does not arrange the tasks, it not only swaps the tasks but also moves the tasks from one machine to other machines in the system. Thus, it reduces the makespan.

#### 4.1.2 Proposed Enhanced Refinery Heuristic

ER heuristic works in two phases. First phase assigns tasks according to Min-min heuristic. Second phase reassigns the tasks that are assigned in the first phase. Reassignment is done by reallocating a task from highest completion time machine to other machines in the system. Tasks are reassigned either by swap or move strategy. Swap strategy retains the same number of tasks while move strategy changes the number of tasks. We describe our proposed ER heuristic in the next section.

**First Phase:** It is very important to select a better initial scheduling solution to achieve minimum makespan. Thus, we have chosen the Min-min heuristic. Min-min heuristic is known as benchmark heuristic and gives the minimum makespan. This heuristic first finds minimum completion time of all unmapped tasks. Next, the task which has minimum completion time is selected and mapped to the machine. Then, the newly mapped task is removed; the process repeats itself until all tasks are mapped.

**Second Phase:** In a distributed environment, some machines are overloaded while other machines are underloaded. We move and swap the tasks between machines to increase the system load balance. Move procedure moves a task from one machine to another machine. Swap procedure swaps tasks between two machines. ER heuristic selects the method which gives minimum makespan. ER heuristic is described in Algorithms 6, 7, and 8. These are described as follows:

ER heuristic (Algorithm-6) first assigns the tasks according to Min-min heuristic (lines 1 to 9). Next, it finds new reassignments using move or swap procedure (lines 13 to 14). Based on the outcome, i.e., whichever method gives minimum

**Algorithm 5** Pseudo Code of ER Heuristic

---

```

1: Input (ETC[task][machine]);
2: for  $T_i = 0 \rightarrow N$  do
3:    $L \leftarrow \text{computeSmallestFinishtime}(\text{ETC});$ 
4:    $k \leftarrow \text{findMin}(L);$ 
5:    $m \leftarrow \text{findMachine}(k);$ 
6:    $\text{assignTask}(k,m);$ 
7:    $\text{remove}(k,\text{ETC});$ 
8:    $\text{updateMachine}(m);$ 
9: end for
10: repeat
11:    $\text{tmp} \leftarrow \text{oldMake} \leftarrow \text{makeTime} \leftarrow \text{computeMakespan}();$ 
12:    $M_m \leftarrow \text{findMachine}(\text{oldMake});$ 
13:    $\text{newMake}' \leftarrow \text{swap}();$ 
14:    $\text{newMake}'' \leftarrow \text{move}();$ 
15:   if  $\text{newMake}' < \text{newMake}''$  then
16:      $\text{swapTask}();$ 
17:      $\text{oldMake} \leftarrow \text{newMake}';$ 
18:   else
19:      $\text{moveTask}();$ 
20:      $\text{oldMake} \leftarrow \text{newMake}'';$ 
21:   end if
22: until  $\text{oldMake} < \text{tmp};$ 

```

---

**Algorithm 6** Pseudo Code of Move Procedure

---

```

1: for all  $T_i \in M_m$  do
2:   for all  $M_j \notin M_m$  do
3:     for  $T_k \in M_j$  do
4:        $\text{comTime}'' \leftarrow \text{computeFinishTime}(M_j);$ 
5:        $\text{comTime}'' \leftarrow \text{comTime}'' + \text{ETC}[T_i][M_j];$ 
6:        $\text{comTime}' \leftarrow \text{makeTime} - \text{ETC}[T_i][M_m];$ 
7:       if  $\text{comTime}'' \leq \text{makeTime}$  then
8:         if  $\text{comTime}' < \text{makeTime}$  then
9:            $\text{makeTime} \leftarrow \text{comTime}';$ 
10:           $\text{task} \leftarrow T_k;$ 
11:           $\text{machine} \leftarrow M_j;$ 
12:        end if
13:      end if
14:    end for
15:  end for
16: end for

```

---



**Algorithm 7** Pseudo Code of Swap Procedure

---

```

1: for all  $T_i \in M_m$  do
2:   for all  $M_j \notin M_m$  do
3:     for  $T_k \in M_j$  do
4:        $comTime'' \leftarrow computeFinishTime(M_j)$ ;
5:        $comTime'' \leftarrow comTime'' - ETC[T_k][M_j] + ETC[T_i][M_j]$ ;
6:        $comTime' \leftarrow makeTime + ETC[T_k][M_m] - ETC[T_i][M_m]$ ;
7:       if  $comTime'' \leq makeTime$  then
8:         if  $comTime' < makeTime$  then
9:            $makeTime \leftarrow comTime'$ ;
10:        end if
11:       end if
12:     end for
13:   end for
14: end for

```

---

makespan is selected, and then the task is reassigned. This process works iteratively, until the makespan changes.

Algorithm-7 describes the move procedure. It finds a task which can be removed from  $M_m$  machine (a machine that has maximum completion time) and be assigned to another machine in the system in order to reduce the makespan. Then, it computes completion time of  $M_m$  machine and other machine. If completion time of  $M_m$  machine and other machine is less than the makespan, then the makespan gets updated. This process keeps information about machine and task, where and which task will be moved.

Algorithm-8 shows the swap procedure. This method works similar to the move procedure. Instead of moving a task, it swaps the tasks between two machines so that same number of tasks are retained on the  $M_m$  machine.

### 4.1.3 Illustrative Example of Enhanced Refinery Heuristic

Consider a sample of ETC matrix which is shown in Fig. 4.1. It has 15 tasks and three machines. ER heuristic first assigns tasks according to Min-min heuristic. Makespan of Min-min heuristic is 921 as shown in Fig. 4.2. Now, we find a new makespan from move or swap procedure. In Iteration-1 swap procedure swaps the task T4 and T8 from machine M1 to M2. This makes makespan 843, (shown in Fig. 4.3). Move procedure (Fig. 4.4) moves a task T3 from machine M2 and

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
M0	694	179	237	391	401	75	593	545	109	35	433	78	1	163	31
M1	604	279	237	469	451	667	75	69	25	52	11	386	271	111	271
M2	594	309	532	157	151	593	223	545	31	18	217	1	631	244	101

Figure 4.1: ETC Matrix

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179				75							1	163	31	449
M1			237				75	69	25		11					417
M2	594			157	151					18		1				921

Figure 4.2: Min-min Heuristic

assigns to machine M0 and makes makespan 840. Move procedure makespan is lesser than swap procedure. Therefore, we choose the move results for next iteration. In Iteration-2, swap procedure swaps the task T2 and T3 from machine M0 to M1 and makes a new makespan 764 (shown in Fig. 4.5). Move procedure moves a task T13 from machine M0 and assigns it to M1. The results are shown in Fig. 4.6. Here, we again choose the move results for next iteration. In Iteration-3, swap does not reduce the makespan. Move procedure further moves a task T9 from machine M2 and assigns it to machine M1. The results are presented in Fig. 4.7. Iteration-4 does not reduce the makespan further. Thus, ER heuristic makespan comes out to be 746 whereas Refinery heuristic makespan is 764.

The time complexity of ER heuristic is similar to Refinery heuristic. The time complexity of first phase is  $O(m \times n^2)$  where  $m$  is the number of machines and  $n$  is the number of tasks. The time complexity of move and swap procedure are  $O(k \times m \times n^2)$  and  $O(k \times m \times n^2)$  respectively. Where  $k$  is the number of iterations when the makespan value change,  $m$  is the number of machines, and  $n$  is the number of tasks. So, the time complexity is maximum of all the procedure which is  $O(k \times m \times n^2)$ .

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179				75							1	163	31	449
M1			237		451		75	69			11					843
M2	594			157					31	18		1				801

Figure 4.3: Iteration-1 of ER Heuristic (Swap Procedure)

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179		391		75							1	163	31	840
M1			237				75	69	25		11					417
M2	594				151					18		1				764

Figure 4.4: Iteration-1 of ER Heuristic (Move Procedure)

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179	237			75							1	163	31	686
M1				469			75	69	25		11					649
M2	594				151					18		1				764

Figure 4.5: Iteration-2 of ER Heuristic (Swap Procedure)

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179		391		75							1		31	677
M1			237				75	69	25		11			111		528
M2	594				151					18		1				764

Figure 4.6: Iteration-2 of ER Heuristic (Move Procedure)

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	CT
M0		179		391		75							1		31	677
M1			237				75	69	25	52	11			111		580
M2	594				151							1				746

Figure 4.7: Iteration-3 of ER Heuristic (Move Procedure)

#### 4.1.4 Experimental Setup

To evaluate and compare the proposed ER heuristic with the other existing algorithms, a Java based simulator has been developed. A similar simulation model is used by (Braun *et al.* 2001), (Sahu & Chaturvedi 2011). Major classes of the simulator are given as follows:

- Main class provides user interface that takes input from the user as number of tasks, number of machines, meta-task size, machines, and tasks heterogeneity.
- Simulator engine class generates ETC matrix and initializes the scheduling engine based on heuristics.
- Scheduling class maps tasks on resources based on heuristics.

To simulate the heterogeneous environment, ETC matrix of size  $N \times M$  is generated. In the actual Grid systems, state estimation is generally done based on the experimental data, application profiling (Hoschek *et al.* 2000) and benchmarking techniques (Wolski *et al.* 1999), (Gong *et al.* 2002). In this experimental testbed, instead of actual task profiling a pre-computed ETC matrix is used.

A row in the ETC matrix contains the expected time to compute for a given job on each resource, whereas a column consists of the expected time to compute of every job on a given resource. Hence, for a task  $T_i$  and a machine  $M_j$ , an entry of the ETC matrix contains the expected time to compute of task  $T_i$  on resource  $M_j$ .

##### 4.1.4.1 Constructing ETC Matrix

Constructing the ETC matrix is a three steps process as given follows:

- Step 1: All necessary initializations are performed. ETC is a two dimensional array of size  $N \times M$  where  $N$  is the total number of jobs and  $M$  is the total number of machines. To construct the ETC matrix, first a baseline vector  $B$  of integer number is generated. Baseline vector  $B$  will contain  $N$  elements.

- Step 2: It explains the creation of Baseline vector  $B$ . To generate the baseline vector  $B$  a uniform random number  $X_b$  is generated such as  $X_b \in [1, U_b]$  ( $U_b$  is the upper bound of the range of possible values of  $B$ ).  $X_b$  is generated repeatedly for  $N$  times and baseline vector  $B$  is constructed such as  $B[i] = X_{bi}$  where  $1 \leq i \leq N$ .
- Step 3: Rows of the ETC matrix are generated from the baseline vector  $B$ . A uniform random number  $X_r$ , known as row multiplier, is generated such as  $X_r \in (1, U_r)$  ( $U_r$  is the upper bound of the range of possible values of the row multiplier,  $X_r$ ).  $M$  different row multipliers are required for a row.  $X_{rij}$  is generated for every element of the ETC matrix in a row and final value of ETC matrix element is generated as  $ETC[i][j] = X_{rij} \times B[i]$ , where  $1 \leq i \leq N$  and  $1 \leq j \leq M$ .

#### 4.1.4.2 Achieving Heterogeneity

The characteristics of the ETC matrix are varied to achieve the heterogeneity. The variation among the execution times of task for a given machine is defined as task heterogeneity. Task heterogeneity is achieved by changing the upper bound of the random numbers within the base line column vector  $B$ . High task heterogeneity is achieved taking  $U_b = 3000$  and low task heterogeneity is achieved by taking  $U_b = 100$ . Similarly, high machine heterogeneity and low machine heterogeneity are achieved by varying the upper bound,  $U_r$  of the random number used to multiply the base line column vector. High machine heterogeneity is represented by taking  $U_r = 1000$  and low machine heterogeneity is achieved by taking  $U_r = 100$  as adopted by (Braun *et al.* 2001), (Bey *et al.* 2010).

Further, the ETC matrix can be categorized, based on the consistencies as consistent, semi-consistent, and inconsistent. Consistent means, if a machine  $M_j$  executes the task  $T_i$  faster than machine  $M_k$  then, it will execute all the jobs faster than  $M_k$ . On the other hand, inconsistent means a machine  $M_j$  can be faster than machine  $M_k$  for some tasks and slower for others. Semi-consistent matrices are

**Table 4.1:** Excerpt from Inconsistent High Heterogeneity of Tasks and Machine

8371	113539	198121	224209
328009	378379	515152	607315
624898	631797	642177	733006
749809	797941	1261081	1275031

**Table 4.2:** Excerpt from Semi High Heterogeneity of Tasks and Machine

1	1742	3805	4005
4439	5006	5073	6965
6965	7008	7408	8009
8706	9877	9877	17284

inconsistent matrices that include a consistent sub matrix of a predefined size. In the semi-consistent ETC matrices used here, 50% of the tasks define a consistent sub-matrix. Based on the above idea (described in section 4.1.4), four categories were proposed for the ETC matrix. These are given as follows:

1. High task heterogeneity and high machine heterogeneity (hi-hi)
2. High task heterogeneity and low machine heterogeneity (hi-lo)
3. Low task heterogeneity and high machine heterogeneity (lo-hi), and
4. Low task heterogeneity and low machine heterogeneity (lo-lo)

Sample  $4 \times 4$  excerpt for inconsistent and semi-consistent matrices are shown in the Table 4.1 and 4.2 respectively. These are taken from the actual matrix of  $512 \times 16$ .

#### 4.1.5 Experimental Results

This section provides the experimental details and results for the proposed scheduling algorithm. ETC matrices are generated using simulation model specified in section 4.1.4.2. The average variation along the rows is referred as the machine heterogeneity and the average variations along the columns are referred as the task heterogeneity and above specified (section 4.1.4.2) values are used. To compare the results with already existing algorithms, we have considered the  $512 \times 16$  matrices as given, i.e. number of jobs ( $N$ ) = 512 and number of machines

( $M$ ) = 16 as adopted by (Braun *et al.* 2001), (Parsa & Entezari-Malekir 2009), (Bey *et al.* 2010).

In order to evaluate the proposed approach, we have implemented the heuristics described in section 4.1.1, and compared our output with Min-min heuristic and Refinery heuristic. The performance of the ER heuristic was evaluated by the average makespan of 100 results on the 100 ETCs generated by the same parameter used by (Bey *et al.* 2010), (Braun *et al.* 2001), (Parsa & Entezari-Malekir 2009).

Fig. 4.8 to 4.11 show the comparison of average makespan of Min-min, Refinery, and ER heuristics. We have shown graphs based on the task and machine heterogeneity. At X-axis, instances are labeled as  $u - yy - zz - x$ .  $u$  means uniform distribution (used in generating the matrix),  $yy$  indicates the heterogeneity of the tasks (hi means high, and lo means low),  $zz$  indicates the heterogeneity of the machines (hi means high, and lo means low), and  $x$  means the type of consistency (c means consistent, i means inconsistent, and s means semi-consistent). At Y-axis, average makespan is shown. From the figures, it is clearly visible that ER and Refinery heuristics performs better than Min-min heuristic in each case. While ER heuristic reduces the makespan over Refinery heuristic by 5%, 12%, 12%, and 24% in lo-lo, lo-hi, hi-lo, and hi-hi consistencies respectively. Here, 5% improvement in the case of lo-lo heterogeneity over Refinery heuristic because there is less number of move and swap occurs. While 24% improvement in the case of hi-hi heterogeneity over Refinery heuristic because there is more number of move and swap occurs. This shows that it is feasible to use ER heuristic. Fig. 4.12 depicts the improvement of ER heuristic over Refinery heuristic with the different consistency. From the Fig. 4.12 it is clearly visible that ER heuristic reduces the makespan by 6%, 15%, and 9% in the case of consistent, semi-consistent, and inconsistent. In the case of inconsistent there is 9% improvement in makespan that is remarkable.

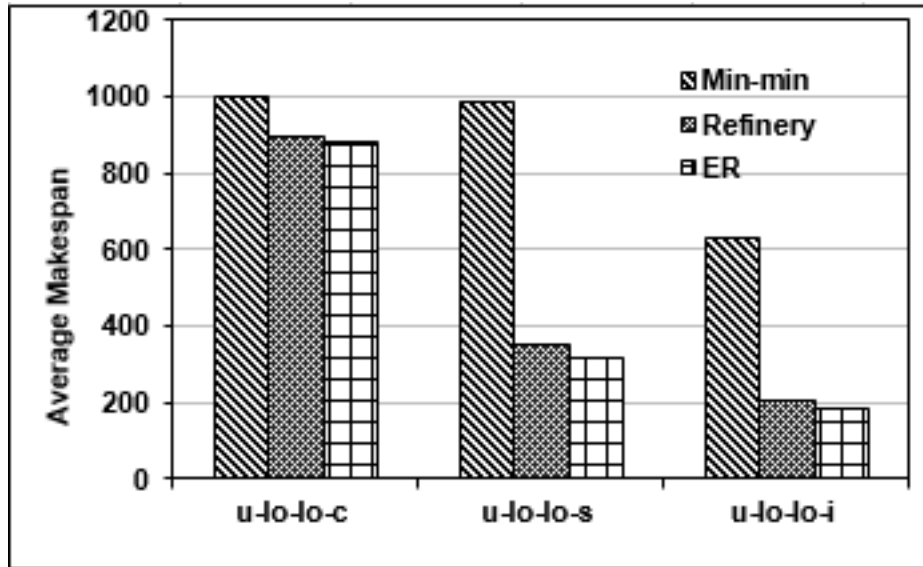


Figure 4.8: Average Makespan of lo-lo Heterogeneity

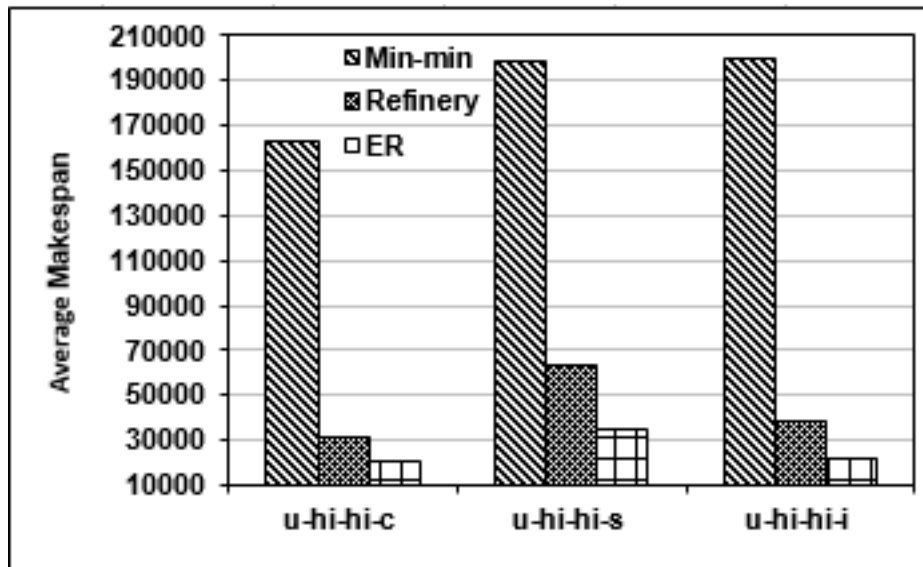


Figure 4.9: Average Makespan of hi-hi Heterogeneity



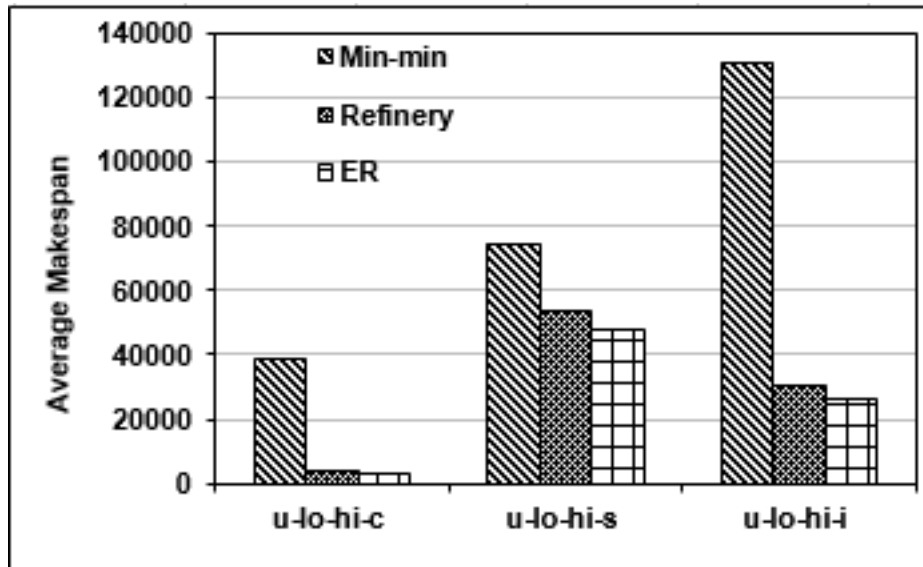


Figure 4.10: Average Makespan of lo-hi Heterogeneity

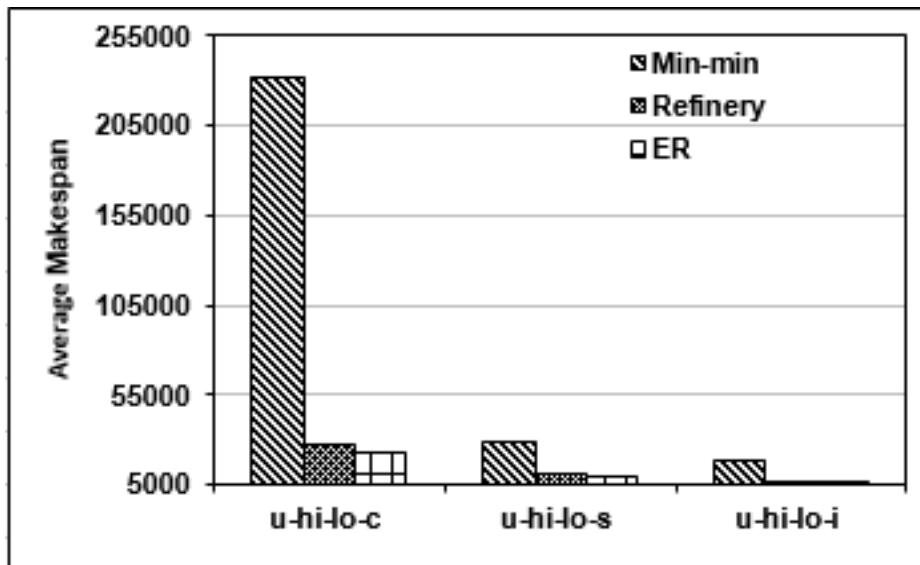


Figure 4.11: Average Makespan of hi-lo Heterogeneity

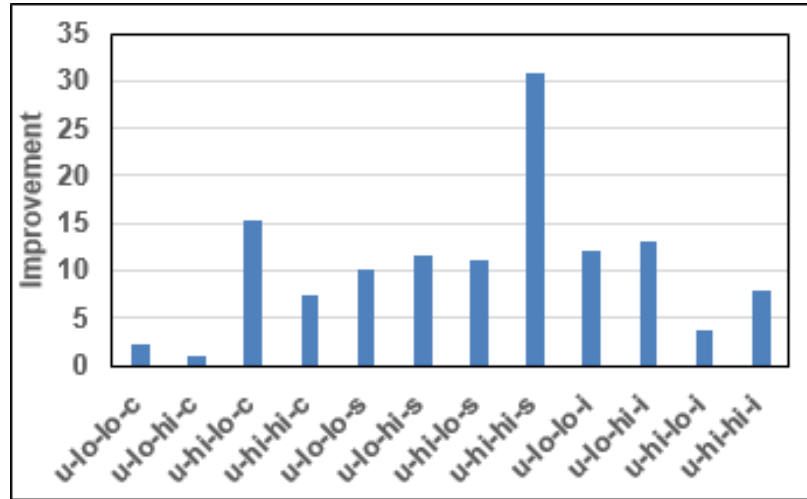


Figure 4.12: Improvement of ER Heuristic Over Refinery Heuristic

## 4.2 Parallel Task Scheduling Algorithm

This section discusses our proposed online parallel task scheduling algorithm on economic Grid. In economic Grid, two parties that get benefit are users and service providers. In order to get efficient and effective resources, user has to pay-per-use, otherwise service provider gives the least priority to the user work. Online task scheduling approach assigns the tasks to machines as task arrives into the system. Existing Parallel task scheduling approaches suffer from processor fragmentation and heterogeneity. Processor fragmentation means sufficient number of processors are available in Grid, but not at single site. Scheduling becomes more challenging when clusters are heterogeneous in computation speed and price.

### 4.2.1 Motivation

Best fit (Huang *et al.* 2007) algorithm selects the resources that leaves the smallest hole at cluster. As a result, this approach increases the load balance and average response time. Fastest fit (Huang *et al.* 2007) algorithm selects the resources that takes the minimum execution time. Result of this algorithm is the least average response time but more load imbalance among the clusters. Adaptive (Shih *et al.* 2013), (Huang *et al.* 2009) algorithms go through the waiting queue before scheduling the current task in order to find which strategy (Fastest

fit or Best fit) will complete the maximum number of tasks in minimum time. (Garg *et al.* 2008) have applied hybrid GA, that first finds the initial scheduling using linear programming without considering the PE requirement of application, then converts this initial assignment into PE requirement of application, if possible, otherwise assigns the task to dummy node for the next iteration. Thus, Existing algorithms that consider both the issues are computationally expensive and are not scalable. We propose a new online parallel task scheduling algorithm that adds cluster cost as one more dimension to the problem. Thus, minimization of cost, time, and processor fragmentation are difficult. We employed TOPSIS algorithm that selects the resource that optimizes all three criteria simultaneously.

#### 4.2.2 System Model

Grid scheduling architecture in Fig. 1.14 consists of following entities:

**Grid Information Server (GIS):** A GIS contains information about all available Grid resources with their computing capacity and cost at which they offer their services to Grid users.

**Service/Resource Providers:** Service providers are resource owners, including clusters, servers, and supercomputers. They are responsible for executing Grid user application. Resource provider provides static information to Grid Information Server (GIS). Static information includes CPU speed in terms of Million Instructions Per Second (MIPS), operating system, the number of PEs, and the usage cost per second.

**Users:** Users have to be registered with GIS and submit their applications to the resource broker for execution on Grid. They also supply information of the QoS parameters.

**Resource Broker:** Resource broker schedules applications for the resource provider. It collects the resources information from the GIS.

This work assumes that all the participants trust and benefit one another by cooperating with one another. It is assumed that service price does not change during the scheduling of applications. We assume an application requires fixed

number of PEs and an application cannot be executed until all the required PEs are available simultaneously. Application of this type of requirement is Synchronous parallel applications or Single Instruction Multiple Data (SIMD).

### 4.2.3 Problem Statement

We first modeled the applications as parallel applications where the application requires more than one computing elements. We proposed Economic Minimum Completion Time (EMCT) scheduling algorithm to select the resources based on trade-off factors that indicates the importance level of cost, time, and PE. To solve this multi-criterion scheduling problem, we used the well known TOPSIS algorithm.

Proposed algorithm considers a Grid environment that consists of a set of resource providers,  $R = \{r_j, r_{j+1}, \dots, r_m\}$ , each resource's available time-slots,  $TS = \{t_k, t_{k+1}, \dots, t_p\}$ , and a set of Parallel applications,  $A = \{a_i, a_{i+1}, \dots, a_n\}$ . Each application is characterized by,  $a_i = (al_i, ap_i, ad_i, as_i)$  where  $al_i$  is the application length in number of instructions that can be estimated using application profiling or benchmarking techniques,  $ap_i$  is the number of Processing Elements (PE) required,  $ad_i$  is application input data in bits, and  $as_i$  is the application submission time.

Each resource characteristic is defined by,  $r_i = (rc_j, rs_j, rp_j, rb_j, rt_j)$ , where the first three parameters are static and others are dynamic. The  $rc_j$  is resource processing cost per second,  $rs_j$  represents resource computational power in terms MIPS of one PE,  $rp_j$  is the number of processors a resource has, to execute a task,  $rb_j$  is resource bandwidth that changes periodically, and  $rt_j$  is the list of available time-slots. Time-slot is characterized by,  $ts_k = (ts_k, tf_k, tp_k)$ , where  $ts_k$  is start time of time slot,  $tf_k$  is finish time of time slot, and  $tp_k$  available PE of time slot. We assume that application  $a_i$  cannot be executed until all the required  $ap_i$  are available simultaneously.

Let  $m$  be the total number of resource providers available for the application  $a_i$ . Here, we assume application is type of rigid; must be processed simultaneously

Table 4.3: Notations

Notation	Definition
$al$	Length of application
$ad$	Input data of application
$ap$	Required PE of application
$as$	Submission time of application
$rc$	Resource cost per second in G\$
$rs$	Resource MIPS
$rp$	Resource PE
$rb$	Resource bandwidth
$rt$	List of time-slots
$ts$	Start time of time-slot
$tf$	Finish time of time-slot
$tp$	Available PE of time-slot

on required number of processors. If, resource providers want to earn more, processor fragmentation should be minimized, so that more number of applications can execute. The used notations are described in Table 4.3.

A lower bound of cost and time of all successful applications can be calculated as minimum cost and minimum time. Value of cost is minimum when application is scheduled on cheapest resources. Value of time is minimum when application is scheduled on fastest resources. Execution time of application  $a_i$  on resource provider  $r_j$  is given by

$$\Psi_{ij} = \frac{al_i}{rs_j} \quad (4.1)$$

Response time of application  $a_i$  on resource provider  $r_j$  in time slot  $t_k$  is given by

$$\alpha_{ij} = tf_{ik} - as_i \quad (4.2)$$

The cost of executing application  $a_i$  on resource provider  $r_j$  is calculated by

$$c_{ij} = rc_j \times ap_i \times \Psi_{ij} \quad (4.3)$$

Transfer time of application  $a_i$  on resource provider  $r_j$  is given as follows:

$$\tau_{ij} = \frac{al_i + ad_i}{rb_j} \quad (4.4)$$

#### 4.2.4 EMCT Scheduling Algorithm

Our proposed EMCT scheduling algorithm is a combination of the MCT and TOPSIS algorithms. MCT scheduling algorithm assigns the task to a machine which would complete the task at the earliest so that all the machines are busy. TOPSIS algorithm is based on the concept that the chosen alternative has the shortest geometric distance from the positive ideal solution and the longest geometric distance from the negative ideal solution. It is a method of compensatory aggregation that compares a set of alternatives by identifying weights for each criterion, normalizing scores for each criterion and calculating the geometric distance between each alternative and an ideal alternative, whichever is the best score in each criterion.

The pseudo code of EMCT scheduling algorithm is given in Algorithm-9. This algorithm uses a matrix that represents decision matrix of TOPSIS algorithm. Matrix is size of  $[m][n]$ , where  $m$  is the number of resources that satisfy the application requirement and  $n$  is the number of criteria. Matrix  $[0][n]$  contains trade-off factor of each criterion.

Whenever a new application arrives in the system, the broker collects information about resources and the application. Steps 3 to 5 assign the trade-off factor of each criterion. Steps 6 to 8 find free time slots from each resource and determine feasible time slot of each resource. A feasible time slot is a slot which has the number of PEs more than or equal to required PEs and start time is equal to or greater than the application submission time. Steps 9 to 11 calculate the response time, cost, and transfer time of the application using equations 4.1, 4.2, 4.3, and 4.4 respectively. Steps 15 to 17 assign total time, processing cost, and available PEs of time slot as an alternate to the matrix. This process is repeated for all the resources. At step 19, TOPSIS process is called that returns the ideal solution as resource. At Step 20, resource broker reserves the resource for an application.

This process is repeated until applications arrive into the system.

---

**Algorithm 8** Pseudo Code of EMCT Scheduling Algorithm
 

---

```

1: while application arrives into system do
2:   get an application  $a_i$ 
3:    $matrix[0][0] \leftarrow a\eta_i$ 
4:    $matrix[0][1] \leftarrow a\delta_i$ 
5:    $matrix[0][2] \leftarrow \gamma_i$ 
6:   for all resource  $r_j \in R$  do
7:     for all time-slot  $t_k \in TS$  do
8:       if ( $tp_k \geq ap_i$ ) then
9:          $\alpha_{ij} \leftarrow tf_{ik} - as_i$ 
10:         $c_{ij} \leftarrow rc_j \times ap_i \times \Psi_{ij}$ 
11:         $\tau_{ij} \leftarrow \frac{al_i + ad_i}{rb_j}$ 
12:        break
13:       end if
14:     end for
15:      $matrix[j][0] \leftarrow \tau_{ij} + \alpha_{ij}$ 
16:      $matrix[j][1] \leftarrow c_{ij}$ 
17:      $matrix[j][2] \leftarrow tp_k - ap_i$ 
18:   end for
19:    $res \leftarrow topsis(matrix)$ 
20:    $res \leftarrow a_i$ 
21: end while

```

---

#### 4.2.5 TOPSIS Algorithm

Input to TOPSIS algorithm is the decision matrix that contains trade-off factor and value of each criterion. We have discussed TOPSIS algorithm in section 3.4.3.

##### 4.2.5.1 Construction of Decision Matrix D

In the context of resource selection, the effect of each criterion cannot be considered alone and should be viewed as a trade-off factor among various criteria. The decision matrix D can be constructed as shown in Table 4.4.

Here  $i$  denotes the alternative resources  $i=1, 2, \dots, m$ ;  $j$  represents the  $j^{th}$  criterion,  $j = 1, 2, \dots, n$  related to  $i^{th}$  cluster, and  $f_{ij}$  is a crisp value indicating the performance value of each resource  $f_i$  with respect to each criterion  $f_j$ .  $w_j$  denotes the trade-off factor of criterion  $j$  and sum of value of all trade-off factors should be  $\sum_{j=1}^n w_j = 1$ .

**Table 4.4:** Decision Matrix D

	$w_j$	$w_{j+1}$	$w_n$
$r_i$	$f_{ij}$	$f_{ij+1}$	$f_{in}$
$r_{i+1}$	$f_{i+1j}$	$f_{i+1j+1}$	$f_{i+1n}$
..	..	..	..
$r_m$	$f_{mj}$	$f_{mj+1}$	$f_{mn}$

**Table 4.5:** Grid Resources

Site Name	PE	MIPS	Price (G\$)
Delhi	100	1140	0.0069
Kolkata	65	1000	0.0032
Madras	252	1200	0.1267
Hyderabad	200	1330	1.856
Bombay	60	1320	0.1424
Pune	54	166	0.0353
Bangalore	265	1176	0.0627
Chennai	20	1140	0.0061
Indore	26	1330	0.1799

#### 4.2.6 Simulation and Evaluation

We simulated our proposed algorithms on GridSim (Buyya *et al.* 2002) tool kit. Resources are modeled according to specifications given in Table 4.5. Resources like number of PEs, MIPS, and prices are also shown in Table 4.5 where the resource's price is not consistent with PE's MIPS. Grid topology is shown in Fig. 4.13 where users are connected with router1 with speed of 1 MBPS. In the simulation we have used 10 resources that are connected with router2. Router1 is connected with router2 with speed 10 MBPS to simulate Grid environment. Different number of applications are generated. Here after, jobs, tasks, and applications are inter changeable. All the resources are simulated as clusters of PE that employ CBF scheduling algorithm and allow advance reservations. CBF scheduling algorithm uses empty spaces present in the waiting queue. The number of CPUs on each resource are chosen such that the demand of CPUs by all applications will always be greater than the total number of free CPUs available on all the resources.

Jobs are modeled according to the workload Lublin model (Lublin & Feitelson 2003). Lublin model is based on logs of different com-



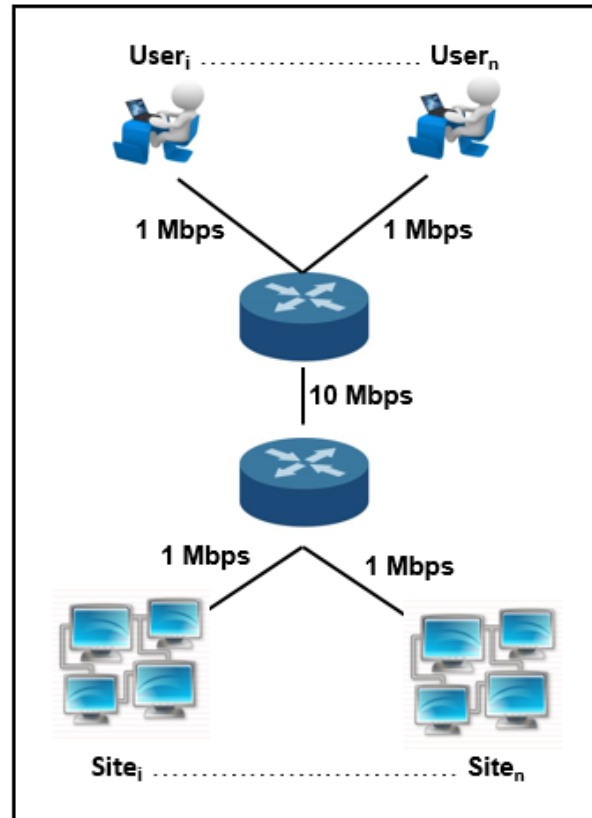


Figure 4.13: Grid Topology

puter system such that San-Diego Supercomputer Center Intel Paragon machine, Los-Alamos National Lab, and Swedish Royal Institute of Technology. This model first apply a logarithmic transformation to the data, due to large range, and then fit it to a novel hyper-gamma distribution function and find the parameters of distribution using iterative expectation maximization algorithm. It derives a function for job length, job run time, job inter-arrival time, and degree of parallelism for batch and interactive jobs. Application run times are generated using a gamma distribution method where mean application length is set and coefficient of variation value is set to 0.9 to test the high variation of application's length.

The performance evaluation of our approach is based on the average cost, average makespan, and average failure. Makespan is the time when the last tasks is finished in given number of applications. Failure is due to the non-availability of required PE of application. Cost is the total money which user has to pay for

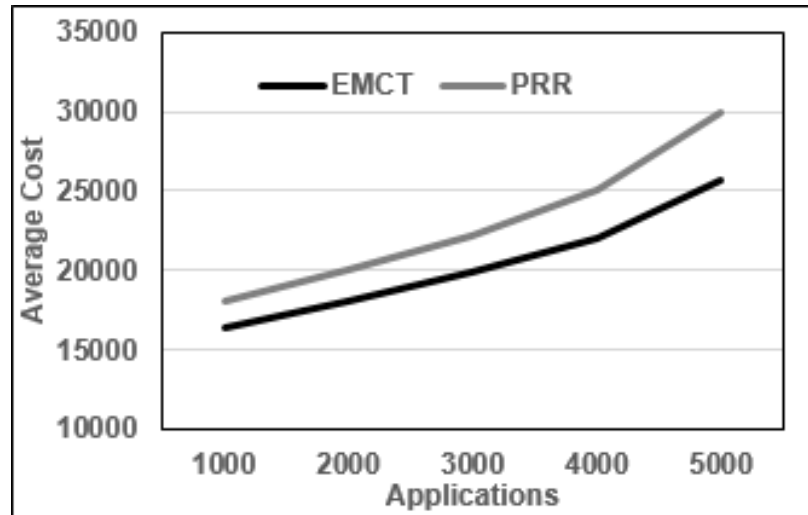


Figure 4.14: Average Cost with Number of Applications

execution of applications. The results presented are averaged out over ten trials with different trade-off factors.

Fig. 4.14 to 4.16 show the average cost, average makespan, and average failure of different number of applications. Cost, time, and PE trade-off are 0.33, 0.33, and 0.34 respectively. In the figures, X-axis corresponds number of applications and Y-axis represents average cost, average makespan, and average failure. We compare our proposed EMCT scheduling algorithm with a Parallel Round Robin (PRR) scheduling algorithm. It is enhancement of Round Robin scheduling algorithm. Round Robin scheduling algorithm schedules a serial independent tasks in circular fashion. PRR scheduling algorithm also schedules tasks in circular fashion but cluster should have required number of PE, other wise, it finds the next cluster that satisfy the task requirement.

Fig. 4.14 shows average cost of PRR scheduling algorithm and EMCT scheduling algorithm. As the number of applications increases cost of both scheduling algorithms is also increases but growth of PRR scheduling algorithm is higher than EMCT scheduling algorithm. Fig. 4.15 displays average makespan with varying number of applications. The average makespan of PRR scheduling algorithm increases greatly, EMCT scheduling algorithm increases mildly. When the number of applications reaches to 5000, the makespan of PRR scheduling algorithm is 45%

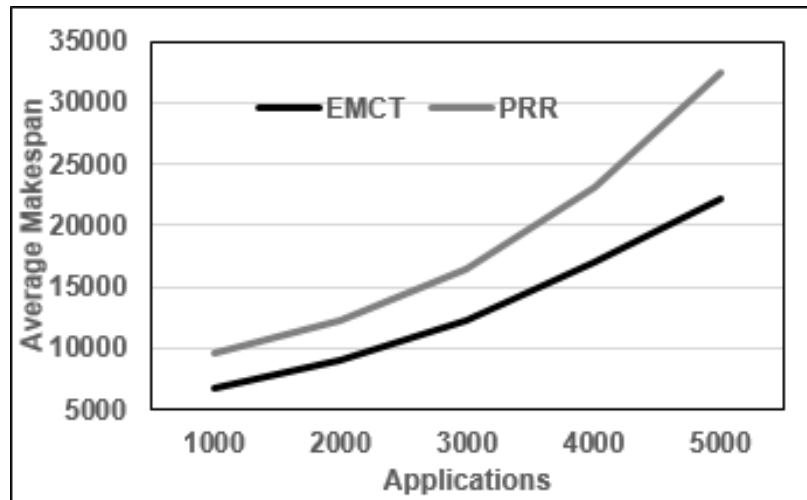


Figure 4.15: Average Makespan with Number of Applications

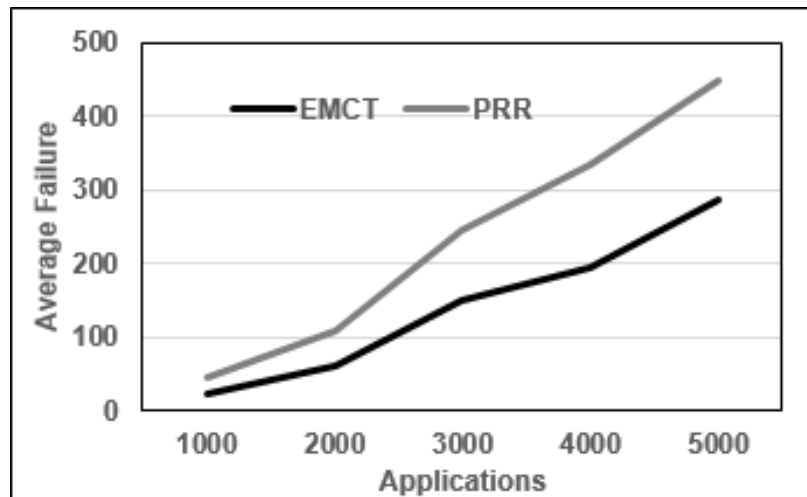


Figure 4.16: Average Failure with Number of Applications

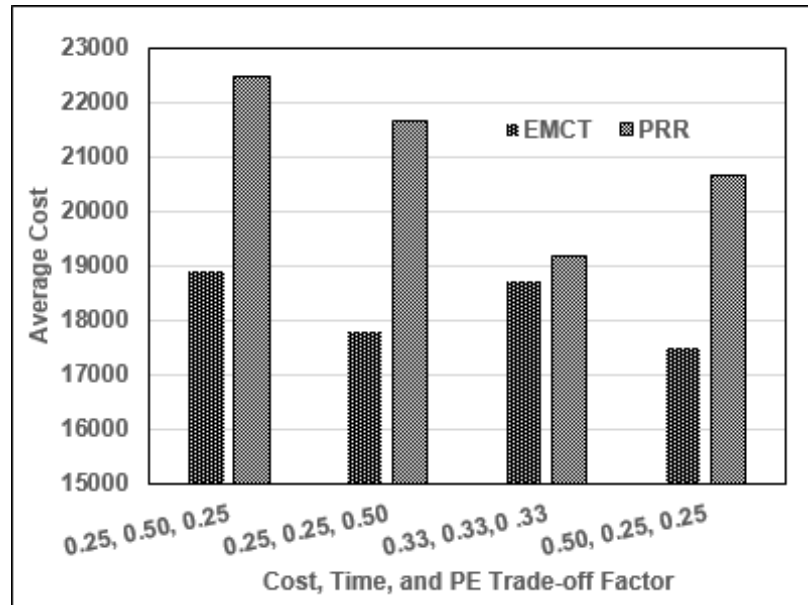


Figure 4.17: Average Cost with Varying Trade-off Factor

higher than the EMCT scheduling algorithm. Fig. 4.16 exhibits the average number of failures with different number of applications. It can be observed that as the number of applications increase, numbers of failures also increase because resources have limited time slots. EMCT scheduling algorithm has average failure is 31% lesser than PRR scheduling algorithm because EMCT scheduling algorithm selects a resource that leaves the smallest fragment free.

In previous experiments, the number of applications submitted is varying, here, we fixed the number of applications to 5000 and analyzed how average makespan, average cost, and average failure change in different trade-off factors, which is shown in Fig. 4.17 to 4.19. In the figures, X-axis corresponds trade-off factor, each trade-off factor is increasing order. As the trade-off factor is increased, objective value is decreased. For example, when trade-off factor of PE is 0.50, number of failure is minimum among all factors. Similar trend is found with other objectives. EMCT performs well when all objective trade-off factors are same. EMCT scheduling algorithm objective value increases mildly, while PRR increase greatly.

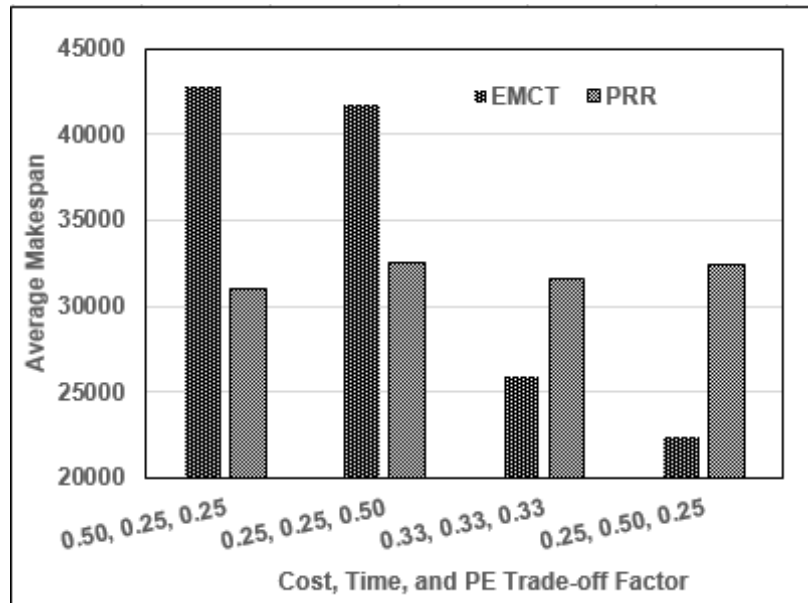


Figure 4.18: Average Makespan with Varying Trade-off Factor

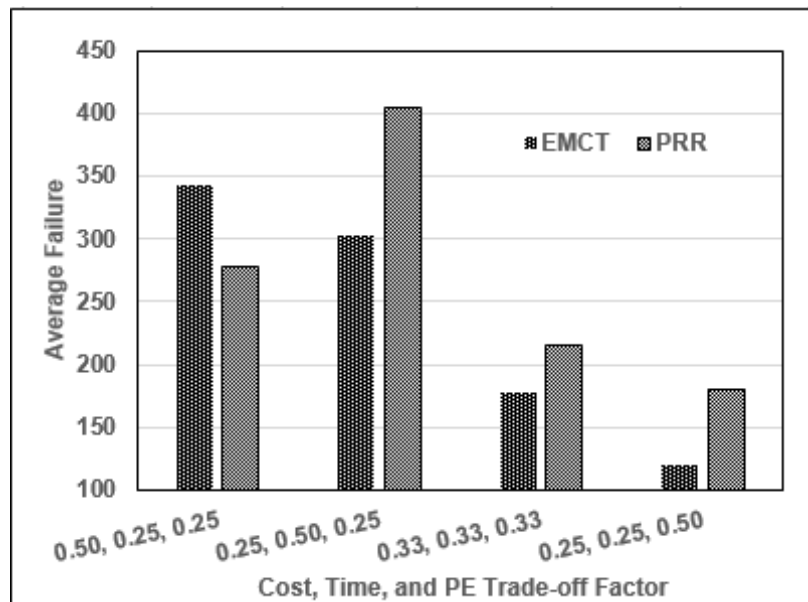


Figure 4.19: Average Failure with Varying Trade-off Factor

### 4.3 Discussion

We have discussed our proposed Enhanced Refinery heuristic and Parallel task scheduling algorithm in this chapter. Enhanced Refinery heuristic works on computational Grid and minimizes the makespan. This heuristic uses the Min-min heuristic as initial algorithm because it is the benchmark algorithm. Enhanced Refinery heuristic not only swaps the tasks but also moves the tasks from one machine to other machines. It axes the makespan by 9% in case of an inconsistency matrix. Parallel task scheduling algorithm works on economic Grid. Objective of Parallel task scheduling algorithm is to minimize the cost, makespan, and processor fragmentation. It is difficult in nature because cost is not consistent with computational speed of resources. Therefore, we used TOPSIS algorithm, that selects the resource based on separation matrix. The GridSim tool and different workload archive are used to demonstrate the efficiency of algorithms. EMCT scheduling algorithm reduces the processor fragmentation by 31% with different number of applications.

## Chapter 5

# Decentralized Task Scheduling

We have developed two Decentralized task scheduling algorithms in which scheduler model interacts among themselves in order to decide which resources should be allocated to the jobs being executed. Our proposed scheduling algorithms are Efficient Dynamic Round Robin scheduling algorithm and Enhanced Sender-initiated scheduling algorithm. Efficient Dynamic Round Robin scheduling algorithm models a scheduling system as a state-transition diagram and replicates a task intuitively. Enhanced Sender-initiated (ESender) scheduling algorithm uses polling information to determine the threshold. Simulation study and a comparison of the results with other similar scheduling algorithms reveal the effectiveness of the of proposed scheduling algorithms.

### 5.1 Motivation

Traditional Round Robin (RR) scheduling algorithm schedules a task to a resource without considering the workload of task and resource, as a result poor performance. In order to improve the performance Multiple work queue (Lee & Zomaya 2006) scheduling algorithm has developed. Multiple work queue scheduling algorithm sorts tasks and hosts, based on task length and initial processing speed of host respectively. After that, tasks are grouped into multiple queue and scheduled from the queue based on host rank. Host rank is computed

based on current performance of node. Dynamic Round Robin (DRR) scheduling algorithm (Lee *et al.* 2006) models a scheduling system as a state-transition diagram and tasks are replicate if waiting queue is empty and processor is free. Replica is used when task requirement and processor processing efficiency is not known. It is used to increase the resource utilization rate or reduce the response time of tasks. We have developed an Efficient Dynamic Round Robin (EDRR) scheduling algorithm that intuitively duplicates a tasks based on current performance of node.

## 5.2 EDRR Scheduling Algorithm

This scheduling algorithm considers a Grid system comprising of distributed computing nodes and a central server for task allocation purpose. The Grid is modeled in the state transition diagram as shown in Fig. 5.1, to be passing through among the four states. The system comprises two queues to store records of the tasks currently in the Grid, namely the waiting queue and the execution queue. The waiting queue comprises of tasks in the Grid, which are yet to be mapped to the machines, while the execution queue contains all the tasks which are currently in execution on at least one of the machines in the Grid.

### 5.2.1 Assumptions

- The model assumes that the tasks arriving in the Grid are atomic (cannot be broken into further sub-tasks) and are independent of one another.
- It is assumed that task transfer cost (It occurs when task is transferred to respective machine) and result collection cost is negligible.
- It is assumed that there is no information available on the workload of the incoming tasks as it is not practically feasible to derive information regarding the same without the services of a full-fledged prediction system.
- The approach assumes that the processing speed of individual computing



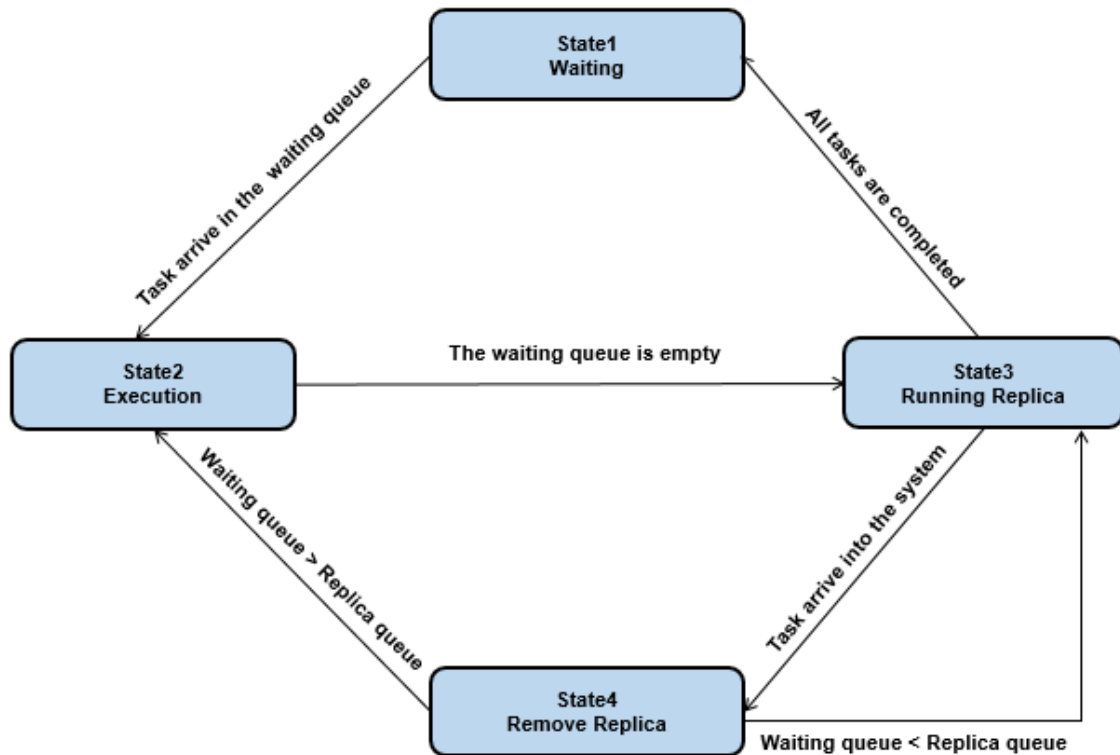


Figure 5.1: EDRR Scheduling Algorithm Grid Model

nodes is available. The initial processing speed of nodes is provided and the processing capacity of machines is updated from time to time on the basis of the last task executed and the time taken.

### 5.2.2 Proposed Solution

Proposed scheduling algorithm is represented using a state transition diagram given in Fig. 5.1 with the Grid occupying one of the four states at any given time. The waiting queue, comprising of tasks waiting to be mapped and executed on their respective machines, is implemented as a First-In-First-Out (FIFO) queue where the task with the earliest arrival time is at the head of the queue and is allocated on idle machine before other tasks waiting in the queue. The execution queue, consisting of tasks currently in execution, is implemented as a circular queue where each task in the queue has a specific order, and no task has the same order as any other task. The execution queue makes use of three pointers to scan

the circular list in a ringed Round-robin fashion. The current pointer is used to point to the task currently having the highest priority in the execution queue. The next pointer is used to point to the task with the second highest priority, which is nothing but the task lined next to the current task, one step in the clockwise direction. The last pointer is used to point to the task with the least priority in the execution queue, which is precisely the task placed behind the current task, a step in the anti-clockwise direction. The following is the description of the four states occupied by the Grid system, during the life time of a task within the Grid system.

#### 5.2.2.1 Waiting State

This state is represented by an idle scheduler that is waiting for tasks to arrive in the Grid. Incoming tasks are lined up in the waiting queue. Both the waiting queue and the execution queue are initially empty. When the number of tasks in the waiting queue becomes more than the threshold value, a transition is made to State II.

#### 5.2.2.2 Execution State

The execution queue is initially empty while the waiting queue contains a number of incoming tasks in the Grid. We maintain a list of idle machines in the system. Initially, all the machines are idle in State II, and hence the list contains all the machines in the Grid. The tasks from the head of the waiting queue are mapped one by one to the machines in the idle list. As soon as a task from the waiting queue gets mapped to a machine, the given machine is subsequently removed from the idle list, while the task is removed from the head of the waiting queue and inserted into the execution queue. The mode of insertion is used to give the highest priority to a task which has been assigned the slowest processor and vice-versa. The task with the highest priority in the execution queue would be the first one to get replicated because it is essentially the task with the slowest processors dedicated to it and hence replicating such a task would lead to a very high probability

of the new machine executing the task before the machines already assigned to it.

### 5.2.2.3 Replica State

The waiting queue is initially empty while the execution queue contains tasks that are currently executing in the Grid. If a machine completes the execution of a task, the processor list of that particular task is referred to, and all the machines dedicated to executing the given task are released and made free while the task is removed; the execution queue and the current, next, and last pointers are updated if required. We update the processing power of the newly freed machine based upon the number of instructions that the machine used for the previous task and the time it took to finish its execution. If the processing speed of the machine is greater than that of the processing speed of the task pointed by the current pointer, then the given machine is required to execute the replica of the task pointed to by the current pointer.

The current, next, and last pointers are also updated as follows. The current pointer becomes the last pointer, the next pointer becomes the current pointer and the next pointer would now be pointing to the task that was one step down in the clockwise direction to the task pointed to by the erstwhile next pointer. Also, if the machine assigned to execute the replica of a task has a processor speed greater than that the processing speed of that task, then the value needs to be updated to the processing speed of the given machine. We also keep track of the number of machines executing replicas and the number of tasks in the waiting queue, this information is exploited in State IV (in section 5.2.2.4). At this point of time, it also checks if there are any other idle machines present in the Grid from the idle list and assigns tasks from the execution queue in the same way as described above, one by one, to these idle machines which are then subsequently removed from the idle list.

However, if a machine is found to have its updated processing speed to be less than that of the processing speed of the current task, then we do not assign the machine to execute the task replica for the simple reason that in all probability,

the task would be accomplished faster on one of the machines already assigned to it as compared to the given machine. The machine name is then inserted at the tail of the list containing the idle machines present in the Grid. There are three scenarios, eventually possible, in State III.

- Case I: All the tasks in the execution queue are successfully completed while the waiting queue is still empty. In this case, we traverse back to State I.
- Case II: All the tasks in the execution queue are successfully completed while the number of tasks in the waiting queue is still less than or equal to the threshold. In this case, we traverse back to State II.
- Case III: The number of tasks in the waiting queue exceeds the threshold before all the tasks in the execution queue could be completed. In this case, we traverse to State IV.

#### 5.2.2.4 Removal of Replica

Both the waiting and the execution queues are initially non-empty. Two scenarios are possible at this point of time.

- Case I: The number of machines executing replicas is less than the number of tasks in the waiting queue.
- Case II: The number of machines executing task replicas is more than the number of tasks in the waiting queue.

The tasks in the execution queue are traversed in an anti-clockwise direction (tasks are assigned in order of processing speed of computing node in a Grid system) one by one, starting from the task pointed to by the last pointer (the least priority task) and if a task has more than one machine allocated to it, the machine at the tail end of the processor list is taken out of the list, freed from the task it was currently executing and assigned the task at the head of the waiting queue (after removing the task from the queue).

In Case I, we stop the traversal as soon as all the tasks in the execution queue are being run on one and only one machine and transit to State II. In Case II, we stop traversing the linked list as soon as all the machines in the waiting queue are assigned a machine, and then subsequently transit to State III.

### 5.2.3 Experimental Setup

Simulation run of our proposed approach consists a Grid network having a fixed number of computing nodes between 200 to 1000 and capacity of resources are between 1 to 200 (in MIPS). The task length is between 1000 to 90000. For simulation purpose, we have considered the four test cases based on the number of tasks and arrival rate of tasks. Cases are following:

- Case I: 6000 tasks and job inter-arrival rate is between 1 to 30.
- Case II: 6000 tasks and job inter-arrival rate is between 1 to 50.
- Case III: 3000 tasks and job inter-arrival rate is between 1 to 30.
- Case IV: 3000 tasks and job inter-arrival rate is between 1 to 50.

We have considered the Average Response Time as the yardstick to test the performance of the EDRR scheduling algorithm against the DRR (Lee *et al.* 2006) scheduling algorithm. Average Response Time is computed as

$$AverageResponseTime = \frac{\sum_{i=1}^n EndTime_i - StartTime_i}{N} \quad (5.1)$$

### 5.2.4 Experimental Results

The results shown are averaged out over thirty trials. Fig. 5.2 to 5.5 show the average response time of four cases that are described in section 5.2.3. In the figures X-axis corresponds to resources and Y-axis represents the average response time. From the Fig. 5.2 to 5.5 it can be observed as the number of resources increases response time decreases. Fig. 5.2 to 5.3 demonstrates average response time of 6000 tasks for case I and II respectively. EDRR scheduling algorithm performs better

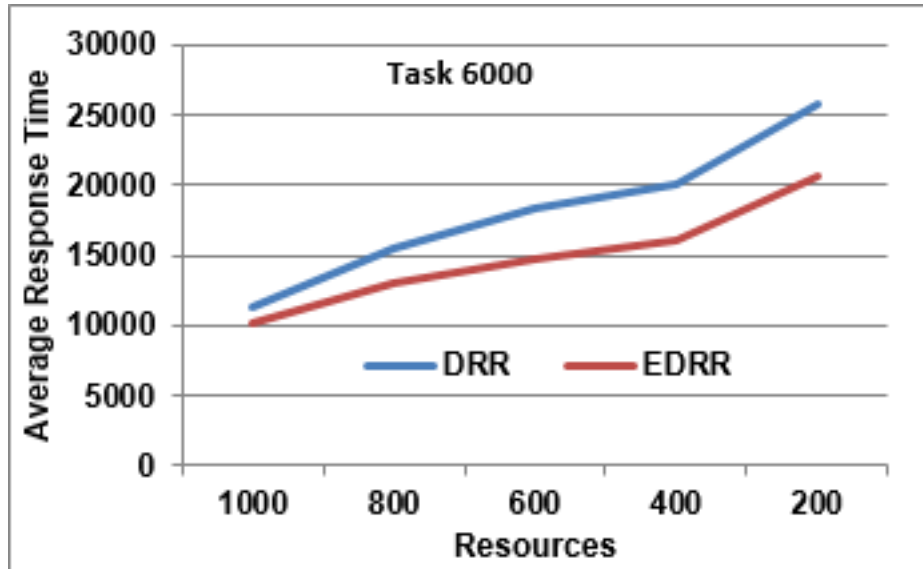


Figure 5.2: Comparison of DRR and EDRR Scheduling Algorithm for Case I

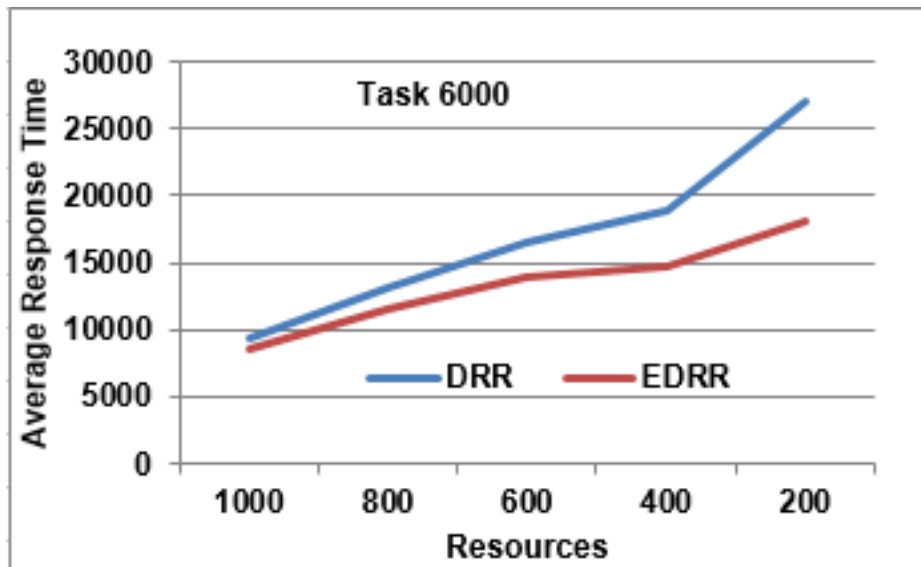


Figure 5.3: Comparison of DRR and EDRR Scheduling Algorithm for Case II

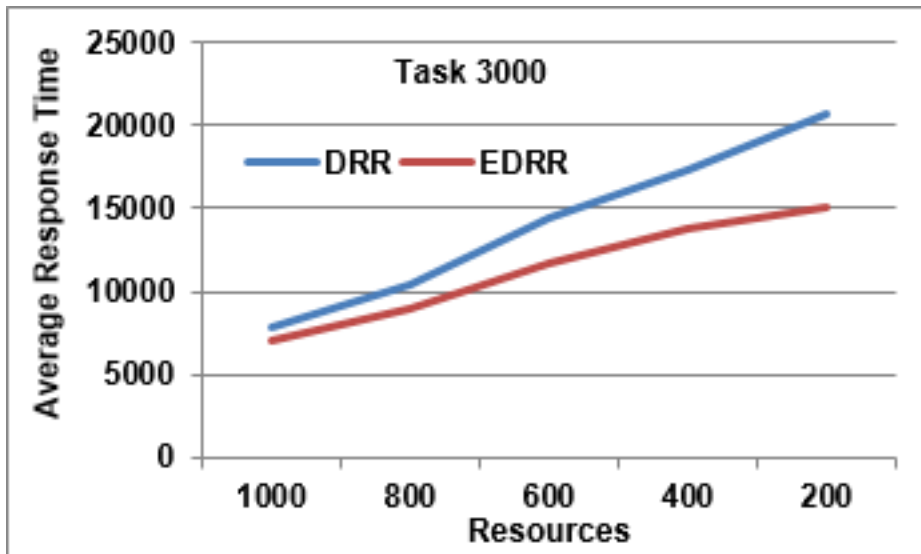


Figure 5.4: Comparison of DRR and EDRR Scheduling Algorithm for Case III

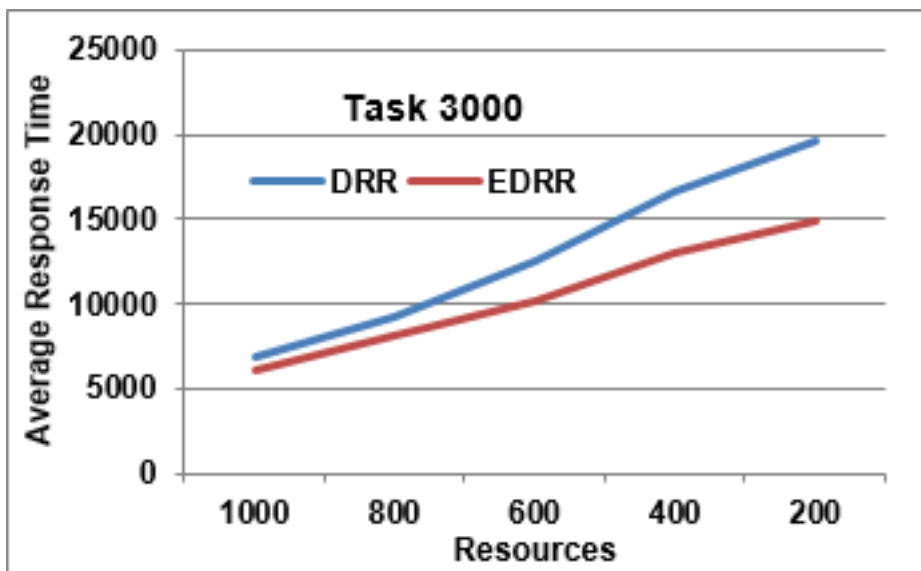


Figure 5.5: Comparison of DRR and EDRR Scheduling Algorithm for Case IV

when job inter-arrival rate is small and less number of resources. Fig. 5.4 to 5.5 show average response time of 3000 tasks for case III and IV respectively. EDDR scheduling algorithm improves overall response time for case I and II by 20% and 13% respectively. This is due to tasks are replicate only if free host capacity is better than available nodes in system.

### 5.3 ESender Scheduling Algorithm

ESender scheduling algorithm eliminates the weakness of Sender-initiated (SI) scheduling algorithm. SI scheduling algorithm works in homogenous system where each node has predefined threshold and an overloaded node polls other nodes in the system. If polling fails, overloaded node gets affected adversely, because polling activity itself increases the system load. Therefore, the performance impact of a query is quite severe at high system load, where most queries fail. ESender scheduling algorithm works in Grid system that is heterogeneous in nature. It reduces the polling activity by broadcasting a message in the system, to increase the queue length that act as a threshold. (Lau *et al.* 2006) have developed an algorithm that works in Functional heterogeneous distributed systems and that transfers multiple tasks. In Functional heterogeneous (Menascé & Almeida 1990) a node can share its workload with only a subset of nodes in the system. Multiple tasks are categorized into different classes based on service demand, code length, and arrival rate. While we have developed an algorithm that does not classify the tasks and works in heterogeneous distributed system (in which overloaded node can share his load to any other nodes in the system). ESender scheduling algorithm is designed to address the above drawbacks. It has the following features:

- It works on heterogeneous system where each node has different threshold, based on common policy.
- At high system load, if sender node does not find the receiver node, it asks for increased threshold, so that sender component will not be deactivated



and, no further communication will take place from other nodes in the system.

The policies on which the proposed scheduling algorithm works are discussed in section 5.3.1 to 5.3.4.

### 5.3.1 Information Policy

Information policy is demand-driven, since polling starts only after a node becomes a sender node. Sender node looks for a receiver node where load can be shared.

### 5.3.2 Transfer Policy

Predefined threshold transfer policy based on CPU queue length is considered a node as a sender node. A node identifies itself as a suitable sender node if accepting the task will increase the node's queue length from threshold  $T$ .

### 5.3.3 Selection Policy

A newly arrived task at the sender node is selected to be transferred.

### 5.3.4 Location Policy

In this policy sender node polls other nodes in the system, if a sender node finds suitable receiver node, it transfers the job to the receiver node. Otherwise, the sender node broadcasts a message to increase the threshold  $T$  by 1%. Thus, all the other nodes will increase threshold  $T$ , no more polling will be taking place in high load. Because if a system has  $n$  nodes in which  $m$  nodes are overloaded, then one of the node polls all the nodes in the system and if it does not find suitable receiver node, it broadcasts a message to increase the threshold  $T$ . Thus, there will be one broadcast message which is equivalent to  $n$  messages. While in SI scheduling algorithm  $n \times m$  messages are taken placed because each overloaded node polls the few nodes in the system, if it does not find the suitable receiver

then process the job locally. Further, it reduces the polling message within the whole system.

The working of ESender scheduling algorithm is shown in Algorithm-10 and flowchart in Fig. 5.6. Here, total  $n$  nodes in the system  $S = (s_i, s_{i+1}, \dots, s_n)$ . When a new job arrives at site  $s_i$ , it checks its queue length; if the queue length is greater than the threshold, it polls all the nodes in the system (lines 3 to 7). During the polling if a sender node finds a receiver node, it transfers the task to receiver node (line 5 to 7). If a node does not find suitable receiver node it broadcasts a message to increase the threshold by 1% (line 10). So other nodes will not perform polling in the system.

The working of SI scheduling algorithm is shown in Algorithm-11. SI scheduling algorithm works similar to ESender scheduling algorithm. In SI scheduling algorithm sender node polls the receiver node till the poll limit. If the sender node does not find receiver node it processes the task locally. As a result, other nodes do not get the benefit of polling activity and at high system load it increases the system load.

---

**Algorithm 9** Pseudo Code of ESender Scheduling Algorithm
 

---

```

1:  $S \in s_i, s_{i+1}, \dots, s_n$ 
2: if  $s_{iq} > t$  then
3:   for  $j = 1 \rightarrow n - 1$  do
4:     poll  $s_j$ 
5:     if  $s_{jq} < t_j$  then
6:       transfer the job
7:     end if
8:   end for
9:   if  $j == n - 1$  then
10:    broadcast message
11:   end if
12: end if

```

---

### 5.3.5 Simulation and Evaluation

The experimental setup is designed as described in section 4.2.6. Ten resources with different MIPS using uniform random distribution between 10 to 20 MIPS are created. Different number of jobs are generated using workload Lublin model

**Algorithm 10** Pseudo Code of Sender-initiated Scheduling Algorithm

---

```

1:  $S \in s_i, s_{i+1}, \dots, s_n$ 
2: if  $s_{iq} > t$  then
3:   for  $j = 0 \rightarrow m$  do
4:     select a node randomly
5:     poll  $s_j$ 
6:     if  $s_{jq} < t$  then
7:       transfer the job
8:     end if
9:   end for
10: end if

```

---

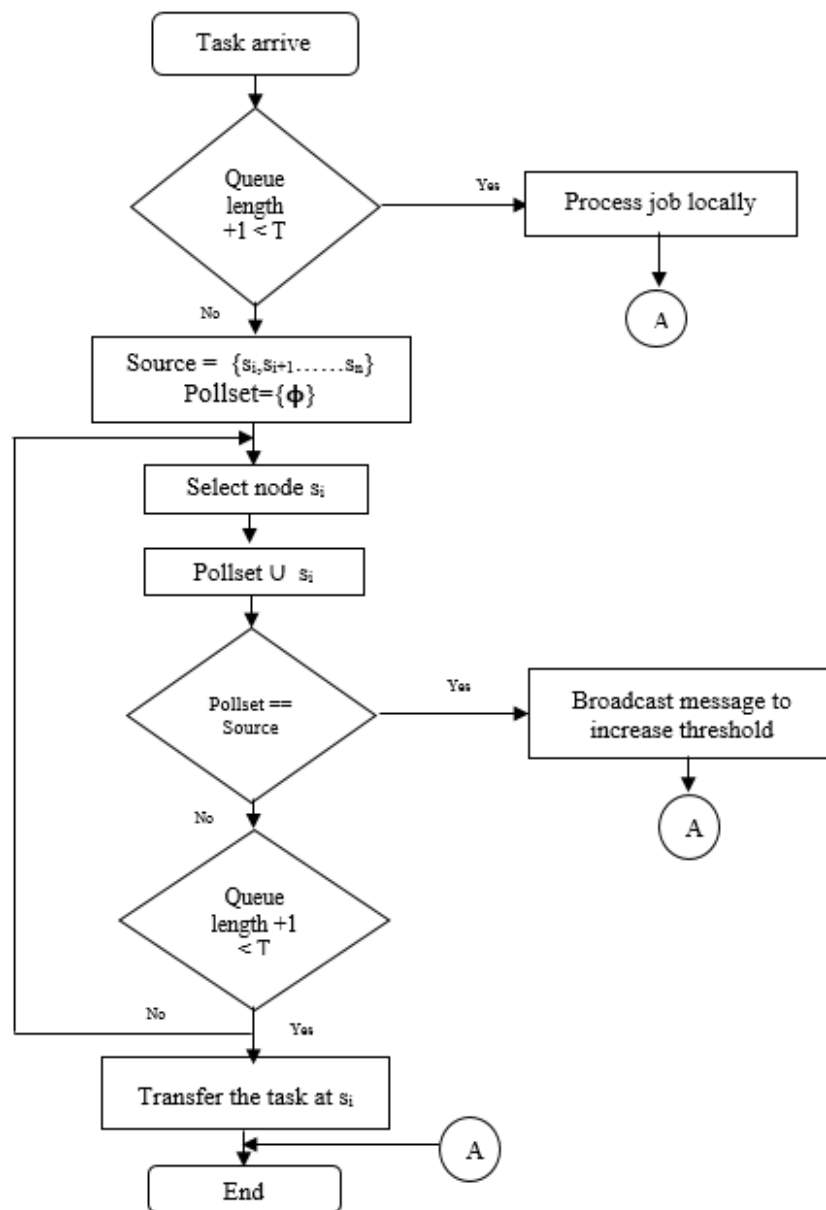


Figure 5.6: Flowchart of ESender Scheduling Algorithm

and assigned to each node uniformly. Each node's threshold is made equal to its number of MIPS multiplied by 5 to observe the appropriate threshold value. If a node has 20 MIPS then its threshold will be 100 tasks. The results presented are averaged out over thirty trials.



Figure 5.7: Comparison of Number of Messages Transfer of SI and ESender Scheduling Algorithm

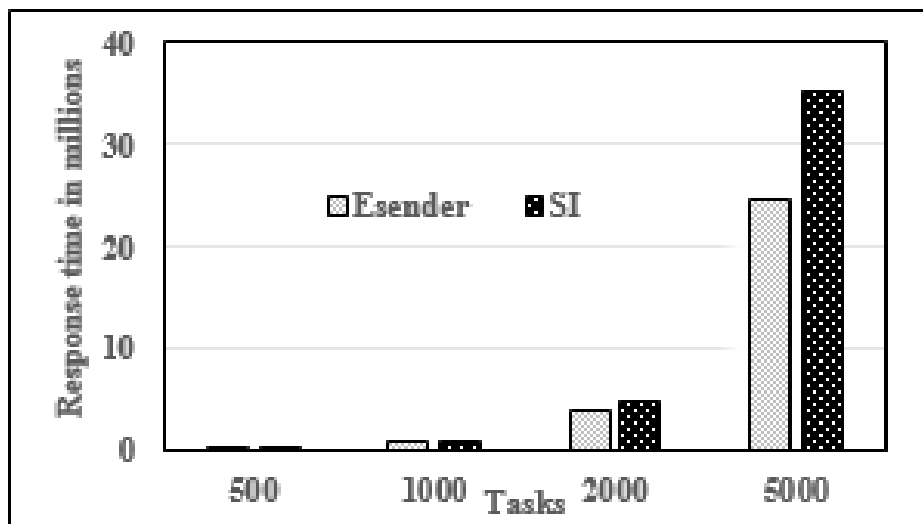


Figure 5.8: Comparison of Turnaround Time with SI of ESender Scheduling Algorithm

Fig. 5.7 shows the number of messages communicated between the sender node and receiver node. X-axis, shows the number of tasks and Y-axis, shows the number of messages. As the number of tasks increases, the number of messages also increases. ESender scheduling algorithm has overall 23% lesser number of

messages than the SI scheduling algorithm. Fig. 5.8 shows the average turnaround time of ESender and SI scheduling algorithm. X-axis, shows the number of tasks and Y-axis, shows the average turnaround time. ESender scheduling algorithm has 12% lesser turnaround time with respect to SI scheduling algorithm.

## 5.4 Discussion

ESender scheduling algorithm eliminates the weakness of SI scheduling algorithm. SI scheduling algorithm works in a distributed system where each node has predefined threshold and an overloaded node polls other nodes in the system. If polling fails, overloaded node affects adversely because polling activity itself increases the system load. Therefore, the performance impact of an inquiry on polling is quite severe at high system load, where most inquiries fail. ESender scheduling algorithm works in Grid system that overcomes this limitation and reduces the polling activity by broadcasting a message in the system, to increase the threshold. ESender scheduling decreases turnaround time by 12% and network overhead by 23%. EDRR scheduling algorithm is an enhancement of existing DRR scheduling algorithm that replicates a task intuitively. This approach is based on exploiting information on processing capability of individual Grid resources and applying replication on tasks assigned to the slowest processors. As a result, Overall response time is improved by 13% when job inter-arrival rate of tasks are large, and 20% improved when job inter-arrival rate of tasks are small.

## Chapter 6

# Conclusions

### 6.1 Conclusions

This thesis presents novel scheduling algorithms for Grid computing systems, which is currently being used by industry for providing a platform for running resource intensive tasks in an efficient manner.

Chapter 1, introduces to these emerging areas and Chapter 2 presents related work. Following are the conclusions that can be drawn:

In Chapter 3, presents dependent task scheduling algorithm on Grid. This scheduling algorithm minimizes three conflict objectives namely makespan, communication cost, and computation cost of execution, using NSGA-II. Various version of NSGA-II has been tested and new Double Hybrid NSGA-II version is introduced by employing pre-selection and memetic operator before and after the NSGA-II algorithm respectively. This approach is 20% minimum objective values than other proposed NSGA-II approaches. Since, multi-objective algorithm generates many solutions, it is nearly impossible to choose the best solution that has minimum cost and time. Therefore, a ranking algorithm is designed to suggest the possible better solutions. Proposed ranking approach obtained 30% to 32% lesser ratio of average makespan and cost than other techniques.

In Chapter 4, we addressed two independent task scheduling algorithms in a Grid. First, offline scheduling algorithm that works on computational Grid,

named as Enhanced Refinery heuristic. Enhanced Refinery heuristic moves and swaps the tasks from overloaded machines to underloaded machines. As a result of our proposed approach, makespan is decreased by makespan by 9% in case of an inconsistent matrix with existing techniques. Second, online parallel task scheduling algorithm works on economic Grid, named as Economic Minimum Completion Time (EMCT) scheduling algorithm. EMCT scheduling algorithm minimizes processor fragmentation, makespan, and cost. EMCT scheduling algorithm is evaluated by an extensive simulation study, which analyzed the best scheduling algorithm to adopt according to different trade-off factors. EMCT reduces overall average failure by 31%.

In Chapter 5, two decentralized scheduling algorithms are introduced. First, Efficient Dynamic Round Robin scheduling algorithm, model a Grid scheduling algorithm as a state transition diagram and duplication candidate task is chosen intuitively to avoid impractical duplication. As a result, Overall response time is improved by 13% when job inter-arrival rate of tasks are large, and 20% improved when job inter-arrival rate of tasks are small. Second, Enhanced Sender-initiated (ESender) scheduling algorithm works in a heterogeneous environment where each node has different transfer policies based resource's MIPS. If a sender node does not find a suitable receiver node, the job is processed locally and a message is broadcast to increase the threshold. ESender scheduling algorithm is compared with Sender-initiated scheduling algorithm based on turnaround time and number of messages. It is shown that ESender scheduling algorithm works better than Sender-initiated scheduling algorithm in case of heterogeneous system. As a result, proposed approach decrease the 12% turnaround time and 23% network overhead.

## 6.2 Summary of Contributions

The following are the contributions of the research carried out as part of this thesis work:

1. Developed a Double Hybrid Non-dominated Sorting Genetic Algorithm-II algorithm for workflow scheduling on Grids.
2. Developed an Independent offline task scheduling that works on computational Grid and reduces makespan of a job submitted to the system.
3. Developed a Parallel task scheduling algorithm that works on economic Grids and minimizes cost, makespan, and processor fragmentation.
4. Developed an Efficient Dynamic Round Robin scheduling algorithm that utilizes free resources effectively.
5. Developed an Enhanced Sender-initiated scheduling algorithm that works on heterogeneous distributed computing environment.

### 6.3 Future Research

The following are the areas that need further research which we plan to pursue in future.

The effectiveness of developed algorithms has been tested on simulators. In future, we would like to test our designed algorithms in real Grid project like Globus or Condor-G. We also would like to design and develop a simulation toolkit for workflow scheduling algorithm for other researchers to experiment their ideas.

As part of our work related to economic Grid scheduling algorithms, we considered fixed computation prices for service providers. We would like to explore two approaches in this context. First, we would like to test a scheduling algorithm that works on variable prices for providers. Second, we would also like to test preemptive task scheduling algorithms, in order to enhance the provider's and user's utility values, where user can move a task from expensive resources to cheaper resources. Similarly, a provider can preempt a task that is less profitable. We plan to work these points in our future research.



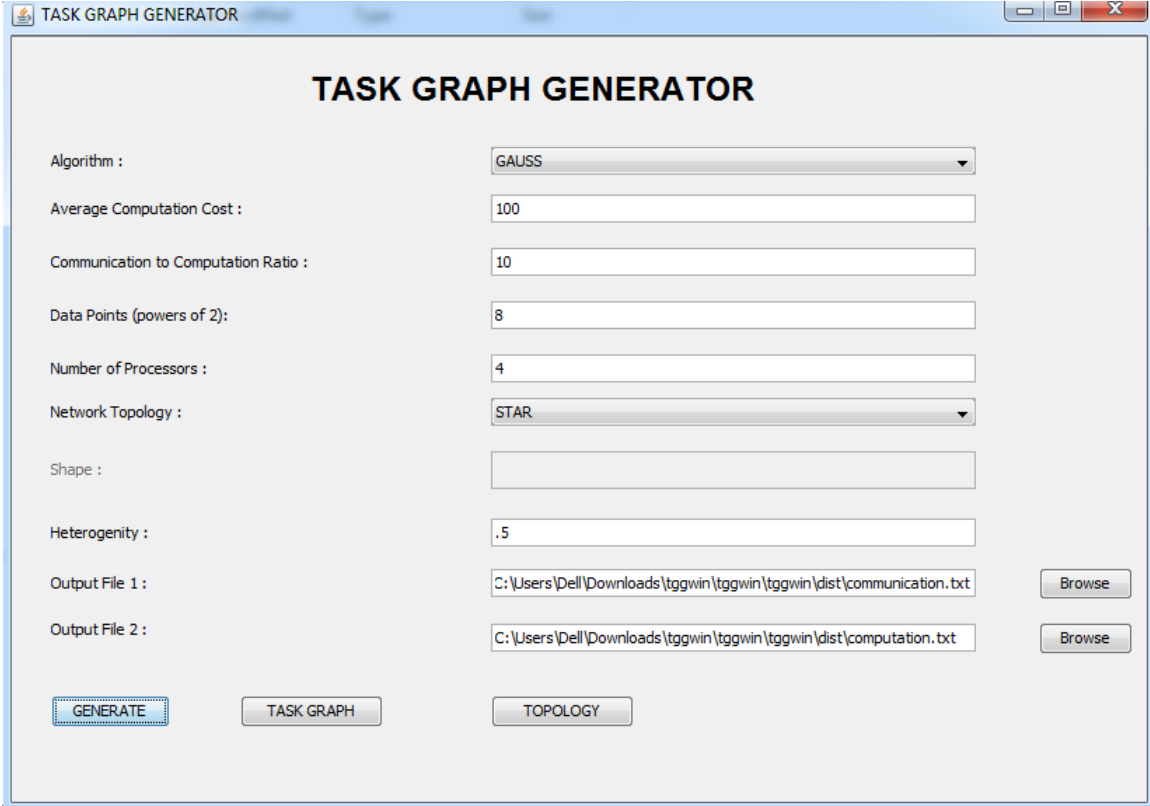
# Appendix A

## Simulator Input-Output

### A.1 DHNSGA-II Input-Output

#### A.1.1 Input

Input screen of Task graph generator as shown in Fig. A.1. It generates graph as shown in Fig. A.2.



The screenshot shows the 'TASK GRAPH GENERATOR' application window. The title bar reads 'TASK GRAPH GENERATOR'. The main content area is titled 'TASK GRAPH GENERATOR' and contains the following input fields and controls:

- Algorithm : GAUSS (dropdown menu)
- Average Computation Cost : 100 (text input)
- Communication to Computation Ratio : 10 (text input)
- Data Points (powers of 2): 8 (text input)
- Number of Processors : 4 (text input)
- Network Topology : STAR (dropdown menu)
- Shape : (empty text input)
- Heterogeneity : .5 (text input)
- Output File 1 : C:\Users\Dell\Downloads\tggwin\tggwin\tggwin\dist\communication.txt (text input) with a 'Browse' button to its right.
- Output File 2 : C:\Users\Dell\Downloads\tggwin\tggwin\tggwin\dist\computation.txt (text input) with a 'Browse' button to its right.

At the bottom of the window, there are three buttons: 'GENERATE' (highlighted with a dashed border), 'TASK GRAPH', and 'TOPOLOGY'.

Figure A.1: Task Graph Generator

It also generates the two output files:

**computation.txt** file keeps the information like Number of processors, Computation cost of each task on each of the processor as shown follows:

```
4
100 97 103 100
97 94 100 97
103 100 106 103
96 93 99 96
104 101 107 104
95 92 98 95jec
105 102 108 105
94 91 97 94
106 103 109 106
93 90 96 93
107 104 110 107
92 89 95 92
108 105 111 108
91 88 94 91
109 106 112 109
90 87 93 90
110 107 113 110
89 86 92 89
111 108 114 111
88 85 91 88
112 109 115 112
87 84 90 87
113 110 116 113
86 83 89 86
114 111 117 114
85 82 88 85
```

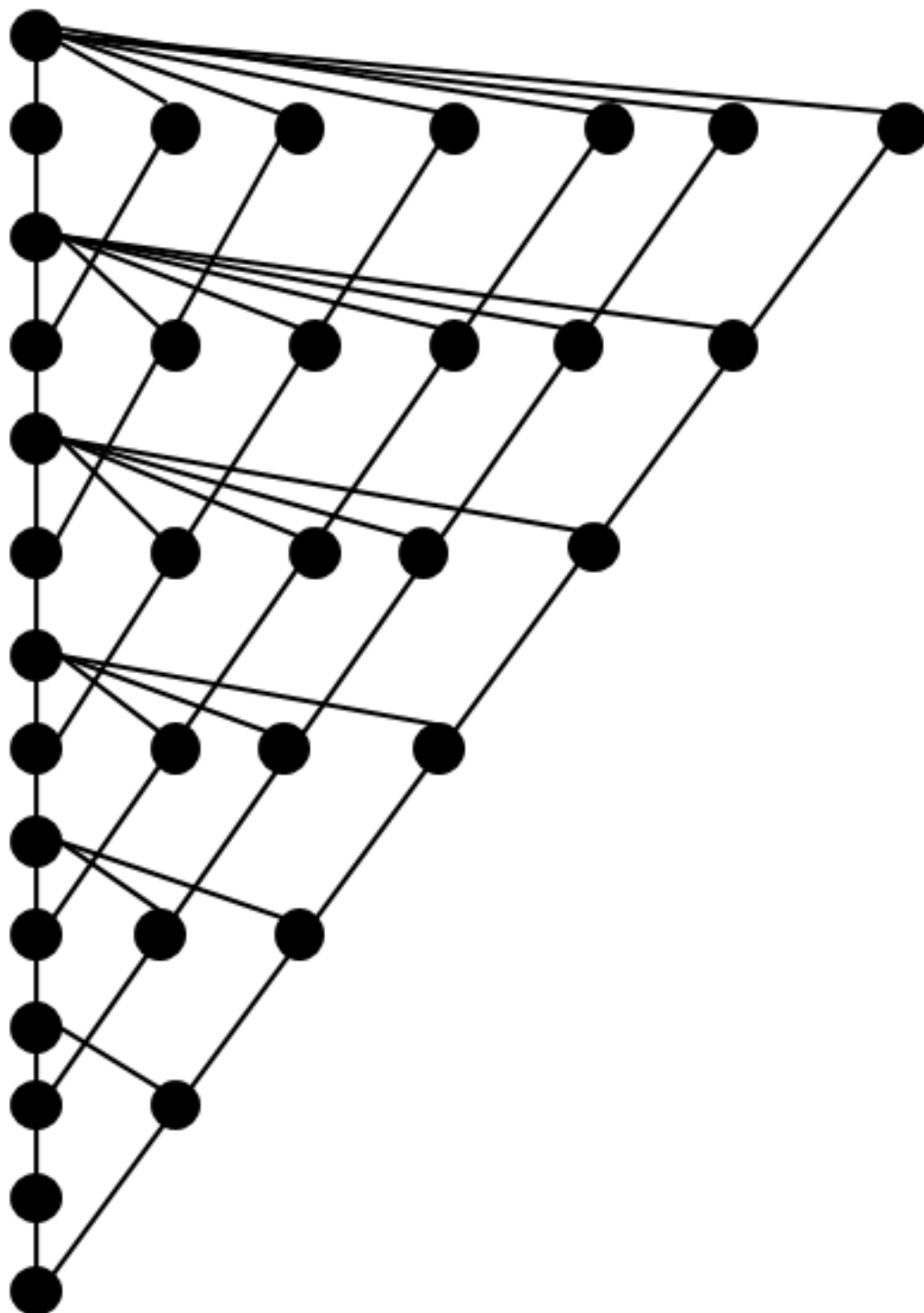


Figure A.2: Gauss Elimination Graph of Matrix Size of 8

115 112 118 115

84 81 87 84

116 113 119 116

83 80 86 83

117 114 120 117

82 79 85 82

118 115 121 118

81 78 84 81

119 116 122 119

**communication.txt file keeps the information like Number of nodes, Number of edges, Communication cost between  $node_i$  and  $node_j$  as shown follows:**

35

55

1 2 500

1 3 497

1 4 503

1 5 496

1 6 504

1 7 495

1 8 505

9 10 494

9 11 506

9 12 493

9 13 507

9 14 492

9 15 508

16 17 491

16 18 509

16 19 490

16 20 510

---

16	21	489
22	23	511
22	24	488
22	25	512
22	26	487
27	28	513
27	29	486
27	30	514
31	32	485
31	33	515
34	35	484
2	9	516
3	10	483
4	11	517
5	12	482
6	13	518
7	14	481
8	15	519
10	16	480
11	17	520
12	18	479
13	19	521
14	20	478
15	21	522
17	22	477
18	23	523
19	24	476
20	25	524
21	26	475
23	27	525

24 28 474  
 25 29 526  
 26 30 473  
 28 31 527  
 29 32 472  
 30 33 528  
 32 34 471  
 33 35 529

### A.1.2 Output

Output of DHNSGA-II is shown as follows:

#### Schedules of GA that minimize makespan

2 2 2 2 2 2 3 2 2 2 1 1 2 1 1 1 2 1 3 1 3 2 2 2 3 2 2 1 1 1 2 2 2 3 2  
 2 2 2 2 2 2 3 2 2 2 1 1 2 1 1 1 2 1 3 1 3 2 2 2 3 2 2 1 1 1 2 2 2 3 2

#### Objective function of GA that minimize makespan

15360.363308021531  
 15360.363308021531

#### Schedules of GA that minimize computation cost

0 0 3 0 0 0 0 0 1 3 1 3 1 1 3 3 2 3 3 3 3 2 2 3 2 2 2 3 2 3 3 3 3 3  
 3 0 0 0 0 0 0 0 0 3 3 0 2 1 0 1 1 3 1 3 3 2 2 3 2 0 2 3 2 3 3 3 3 3 3

#### Objective function of GA that minimize computation cost

6395.428264645327  
 24971.270679300225

#### Schedules of GA that minimize communication cost

1 1 1 3 1 2 1 3 1 3 3 0 3 3 2 3 2 3 1 3 0 2 3 2 0 0 3 3 3 3 2 2 2 3 3  
 3 0 1 3 1 2 1 3 1 3 3 0 3 3 2 3 2 3 1 3 0 2 3 2 0 0 3 3 3 3 2 2 2 3 3

#### Objective function of GA that minimize communication cost

4493.0  
 5183.0

#### Schedules of NSGA-II\*

```

00001000000030000010100201001100000
00001000000010000000100200020100000
00001000000010000000100000020300000
00001000000030000010100201001100000
00001000000010000000100201020100000
000010000000100000001001000000300000
00001000001010000010100000020100000
000010000000100000001002000000300000
0000100000001000001100201000100000
000010000000100000001001000000300000
00001000000010000000100000020300000
000010000000100000001001001000100000

```

#### Objective functions of NSGA-II\*

7372.514792519578 3069.0 17776.482640231785

6387.660019255511 3125.0 17923.23906663212

6078.764435961059 3120.0 19892.566069019307

7372.514792519578 3069.0 17776.482640231785

6712.636979751509 3125.0 16538.0884269168

5850.665742029125 3995.0 21238.425553700028

6647.593362388683 3384.0 17862.035933766118

6091.774485702155 3069.0 19625.44622742642

6412.5116432381255 2947.0 20200.817924373947

5850.665742029125 3995.0 21238.425553700028

6078.764435961059 3120.0 19892.566069019307

5902.446661076887 3992.0 20123.26542820139

#### Schedules of NSGA-II\*\*

3333133333333131333311113111111111

3333133333333131333311101111111111

33331333333133331333313113111111111

3333133333333131333311101111111111

3 3 3 3 1 3 3 3 3 3 3 1 3 3 3 3 1 3 3 3 3 1 3 3 3 3 1 1 1 1 1 1 1 1 1  
 3 3 3 3 1 3 3 3 3 3 3 3 3 3 1 3 1 3 3 3 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1  
 3 3 3 1 1 3 3 1 3 3 3 3 3 3 3 1 1 3 3 3 3 1 3 3 3 3 1 1 1 1 1 3 3 3 3  
 3 3 3 3 1 3 3 3 3 3 3 3 3 3 1 3 1 3 3 3 3 1 3 3 3 1 1 1 1 1 1 1 1 1 3  
 3 3 3 3 1 3 3 3 3 3 3 1 3 3 3 3 1 3 3 3 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1  
 3 3 3 3 1 3 3 3 3 3 3 1 3 3 3 3 1 3 3 3 3 1 3 3 3 1 1 1 1 1 1 1 1 1 1

**Objective functions of NSGA-II\*\***

8227.092115293168 2573.0 19341.56496496086  
 9233.156745353292 2684.0 14717.173292824149  
 7907.7267505158125 2897.0 17617.986697948025  
 9233.156745353292 2684.0 14717.173292824149  
 7421.594844297847 2897.0 25021.46955822536  
 8227.092115293168 2573.0 19341.56496496086  
 6904.943173412531 4254.0 58322.514669971955  
 7378.3585110191025 3387.0 30852.506600115335  
 8183.855782014423 2666.0 15404.738639026093  
 7626.511091572644 2897.0 21415.9734052703

**Schedules of NSGA-II\*\*\***

3 3 3 2 3 3 1 0 3 1 1 2 3 1 0 1 1 2 2 0 1 1 0 0 0 3 0 0 0 0 0 0 0 0 0  
 3 3 3 2 3 3 1 0 3 1 1 2 3 1 0 1 1 3 1 1 0 1 0 1 0 3 0 0 0 0 0 0 0 0 0  
 0 0 2 0 2 0 2 2 0 2 0 2 0 2 2 2 2 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0  
 3 3 3 2 3 3 1 0 3 1 1 2 3 1 0 1 1 2 2 0 1 1 0 0 0 3 0 0 0 0 0 0 0 0 0  
 2 0 0 0 2 0 2 2 0 2 0 2 0 2 2 2 2 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0  
 3 3 3 2 3 3 1 0 3 1 1 2 3 1 0 1 1 2 2 0 1 1 0 0 0 3 0 0 0 0 0 0 0 0 0  
 2 2 2 2 2 0 2 2 0 2 0 2 0 2 2 2 2 2 0 0 0 0 0 0 0 3 2 0 0 0 0 0 0 2 0  
 0 0 2 0 2 0 2 2 0 2 0 2 0 2 2 2 2 0 2 0 0 0 0 0 0 3 2 0 0 0 0 0 0 0 0  
 0 0 0 1 2 2 0 2 0 1 1 0 0 0 0 1 1 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0  
 0 0 2 0 2 0 2 2 0 2 0 2 0 2 0 0 0 2 0 0 0 0 2 2 0 3 2 2 2 2 2 0 0 0 0  
 2 0 0 0 2 0 2 2 0 2 0 2 0 0 0 2 2 2 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0  
 0 0 0 1 2 2 0 2 0 1 1 0 0 0 0 1 1 0 2 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0



33323310311231011220110003000000000  
33323310311231011311010103000000000  
33323310311231011220110003000000000  
00012202011000011020000003000000000  
00012202011000011020000003000000000  
20002022020200022200000003000000000  
00202022020202000200002203222220000  
00202022020202222020000003000000000  
20002022020200022200000003000000000  
33323310311231011220110003000000000  
00202022020202222020000003000000020  
2222200020000000200000003222200000  
22222022020200000020000003000000000  
22222022020202222020000003000000020  
20002022020200022200000003000000000  
33323310311231011311010103000000000  
00202022020202222020000003000000020  
20002022020200022200000003000000000  
22222022020202222020000203000000020  
33323310311231011220110003000000000  
00202022020202000200002203222220000  
33323310311231011311010003000000000  
00002000020020000020000003000000000  
00202022020202222020000003200000000  
00012202011000011020000003000000000  
33323310311231011220110003000000000  
0020202202020222200000003000000020

**Objective functions of NSGA-II\*\*\***

8720.755821795 2904.0 47279.55018336183

10298.929072719906 2928.0 40251.74725213821

570.0372954838606 3028.0 33846.443390455206  
8720.755821795 2904.0 47279.55018336183  
570.2905965079191 3531.0 32591.699574589427  
8720.755821795 2904.0 47279.55018336183  
531.1141318757 4895.0 39722.63766142977  
560.4288953890077 3415.0 40874.7951920279  
4377.8869289865415 3010.0 43970.31678979393  
538.1050880552823 3556.0 39195.72865311218  
588.8319273001357 3531.0 27585.530545026446  
4377.8869289865415 3010.0 43970.31678979393  
8720.755821795 2904.0 47279.55018336183  
10298.929072719906 2928.0 40251.74725213821  
8720.755821795 2904.0 47279.55018336183  
4377.8869289865415 3010.0 43970.31678979393  
4377.8869289865415 3010.0 43970.31678979393  
588.8319273001357 3531.0 27585.530545026446  
538.1050880552823 3556.0 39195.72865311218  
570.0372954838606 3028.0 33846.443390455206  
588.8319273001357 3531.0 27585.530545026446  
8720.755821795 2904.0 47279.55018336183  
563.2996403283366 3332.0 38233.69597045869  
563.9751097258259 4878.0 24282.182148953645  
578.9702261812243 3634.0 21480.712058537953  
538.8649911274576 3854.0 32775.45102396472  
588.8319273001357 3531.0 27585.530545026446  
10298.929072719906 2928.0 40251.74725213821  
563.2996403283366 3332.0 38233.69597045869  
588.8319273001357 3531.0 27585.530545026446  
531.705167598503 4677.0 37203.71016225587  
8720.755821795 2904.0 47279.55018336183

## A. SIMULATOR INPUT-OUTPUT A.2 Enhanced Refinery (ER) Heuristic Input-Output

538.1050880552823 3556.0 39195.72865311218  
9651.95302378372 2928.0 42336.41041448227  
627.2484888601795 3112.0 21694.20833045069  
560.4288953890077 3415.0 40874.7951920279  
4377.8869289865415 3010.0 43970.31678979393  
8720.755821795 2904.0 47279.55018336183  
565.157181171432 3332.0 38152.53080635103

### **A.2 Enhanced Refinery (ER) Heuristic Input-Output**

#### **A.2.1 Input**

This algorithm takes number of tasks, resources, Task Heterogeneity, and Machine Heterogeneity as input

Number of resources = 3, Number of tasks = 15, Task Heterogeneity = High, and Machine Heterogeneity = High

#### **Expected Time to Complete**

694	604	594
179	279	309
237	237	532
391	469	157
401	451	151
75	667	593
593	75	223
545	69	545
109	25	31
35	52	18
433	11	217
78	386	1
1	271	631
163	111	244

31 271 101

### A.2.2 Output

The following section shows the working of different heuristic.

#### **Min-min Heuristic**

Assigning task 11 to machine 2. Completion time = 1  
Assigning task 12 to machine 0. Completion time = 1  
Assigning task 10 to machine 1. Completion time = 11  
Assigning task 9 to machine 2. Completion time = 19  
Assigning task 14 to machine 0. Completion time = 32  
Assigning task 8 to machine 1. Completion time = 36  
Assigning task 7 to machine 1. Completion time = 105  
Assigning task 5 to machine 0. Completion time = 107  
Assigning task 4 to machine 2. Completion time = 170  
Assigning task 6 to machine 1. Completion time = 180  
Assigning task 13 to machine 0. Completion time = 270  
Assigning task 3 to machine 2. Completion time = 327  
Assigning task 2 to machine 1. Completion time = 417  
Assigning task 1 to machine 0. Completion time = 449  
Assigning task 0 to machine 2. Completion time = 921

**Makespan = 921, Strategy: Min-min**

#### **Enhanced Refinery Heuristic**

##### **First Phase**

Assigning task 11 to machine 2. Completion time = 1  
Assigning task 12 to machine 0. Completion time = 1  
Assigning task 10 to machine 1. Completion time = 11  
Assigning task 9 to machine 2. Completion time = 19  
Assigning task 14 to machine 0. Completion time = 32

## A. SIMULATOR INPUT-OUTPUT A.2 Enhanced Refinery (ER) Heuristic Input-Output

Assigning task 8 to machine 1. Completion time = 36  
Assigning task 7 to machine 1. Completion time = 105  
Assigning task 5 to machine 0. Completion time = 107  
Assigning task 4 to machine 2. Completion time = 170  
Assigning task 6 to machine 1. Completion time = 180  
Assigning task 13 to machine 0. Completion time = 270  
Assigning task 3 to machine 2. Completion time = 327  
Assigning task 2 to machine 1. Completion time = 417  
Assigning task 1 to machine 0. Completion time = 449  
Assigning task 0 to machine 2. Completion time = 921

### **Second Phase**

new makespan from move method = 840.0 Task = 3 toMachine = 0  
new makespan from swap method = 843.0 Task = 4 to Task = 8  
move method has less makespan  
Lowest Makespan1 =840.0 fromTask =3 fromIndex =3 toTask =3 toMachine =0  
new makespan from move method = 677.0 Task = 13 toMachine = 1  
new makespan from swap method = 686.0 Task = 3 to Task = 2  
move method has less makespan  
Lowest Makespan1 =677.0 fromTask =13 fromIndex =3 toTask =13 toMachine  
=1  
new makespan from move method = 746.0 Task = 9 toMachine = 0  
new makespan from swap method = 764.0 Task = -1 to Task = -1  
move method has less makespan  
Lowest Makespan1 =746.0 fromTask =9 fromIndex =1 toTask =9 toMachine =0  
new makespan from move method = 746.0 Task = -1 toMachine = -1  
new makespan from swap method = 746.0 Task = -1 to Task = -1  
swap method has less makespan  
Lowest Makespan1 =746.0 fromTask =-1 fromIndex =-1 toTask =-1 toMachine  
=-1

**Final schedule**

Assigning task 11 to machine 2. Completion time = 1  
Assigning task 12 to machine 0. Completion time = 1  
Assigning task 10 to machine 1. Completion time = 11  
Assigning task 9 to machine 0. Completion time = 36  
Assigning task 14 to machine 0. Completion time = 67  
Assigning task 8 to machine 1. Completion time = 36  
Assigning task 7 to machine 1. Completion time = 105  
Assigning task 5 to machine 0. Completion time = 142  
Assigning task 4 to machine 2. Completion time = 152  
Assigning task 6 to machine 1. Completion time = 180  
Assigning task 13 to machine 1. Completion time = 291  
Assigning task 3 to machine 0. Completion time = 533  
Assigning task 2 to machine 1. Completion time = 528  
Assigning task 1 to machine 0. Completion time = 712  
Assigning task 0 to machine 2. Completion time = 746  
**Makespan = 746, Strategy: ER**

**Refinery Heuristic**

The following section shows the working of Refinery heuristic.

**First phase**

Assigning task 11 to machine 2. Completion time = 1  
Assigning task 12 to machine 0. Completion time = 1  
Assigning task 10 to machine 1. Completion time = 11  
Assigning task 9 to machine 2. Completion time = 19  
Assigning task 14 to machine 0. Completion time = 32  
Assigning task 8 to machine 1. Completion time = 36  
Assigning task 7 to machine 1. Completion time = 105

## A. SIMULATOR INPUT-OUTPUT A.2 Enhanced Refinery (ER) Heuristic Input-Output

Assigning task 5 to machine 0. Completion time = 107  
Assigning task 4 to machine 2. Completion time = 170  
Assigning task 6 to machine 1. Completion time = 180  
Assigning task 13 to machine 0. Completion time = 270  
Assigning task 3 to machine 2. Completion time = 327  
Assigning task 2 to machine 1. Completion time = 417  
Assigning task 1 to machine 0. Completion time = 449  
Assigning task 0 to machine 2. Completion time = 921

### **Second Phase**

makespan: 921.0 At machine :2  
new makespan =843.0 firstTask = 4 secondTask =8 toMachine =1 fromIndex 2  
toIndex 1  
makespan: 843.0 At machine :1  
new makespan =671.0 firstTask = 4 secondTask =1 toMachine =0 fromIndex 1  
toIndex 4  
makespan: 801.0 At machine :2  
new makespan =795.0 firstTask = 3 secondTask =4 toMachine =0 fromIndex 3  
toIndex 4  
makespan: 795.0 At machine :2

### **Final schedule**

Assigning task 11 to machine 2  
Assigning task 12 to machine 0  
Assigning task 10 to machine 1  
Assigning task 9 to machine 2  
Assigning task 14 to machine 0  
Assigning task 1 to machine 1  
Assigning task 7 to machine 1  
Assigning task 5 to machine 0

## A. SIMULATOR INPUT-OUTPUT A.2 Enhanced Refinery (ER) Heuristic Input-Output

Assigning task 8 to machine 2

Assigning task 6 to machine 1

Assigning task 13 to machine 0

Assigning task 4 to machine 2

Assigning task 2 to machine 1

Assigning task 3 to machine 0

Assigning task 0 to machine 2

**Makespan = 795, Strategy: Refinery**



# References

- [Abi 2014] *Create, Manage and Govern your Cloud*. <http://www.abiquo.com/>, 2014.
- [Aggarwal *et al.* 2005] M. Aggarwal, R. D. Kent and A. Ngom. *Genetic Algorithm Based Scheduler for Computational Grids*. In Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications, pages 209 – 215, 2005.
- [Aka 2014] *Any Experience. Any Device. Anywhere*. <http://www.akamai.com/>, 2014.
- [Almond & Snelling 1999] J. Almond and D. Snelling. *UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce*. Future Generation Computer Systems, NH-Elsevier, vol. 15, no. 5 - 6, pages 539 – 548, 1999.
- [Ama 2014] *Amazon*. <http://aws.amazon.com/>, 2014.
- [Amudha & Dhivyaprabha 2011] T. Amudha and T. T. Dhivyaprabha. *QoS Priority Based Scheduling Algorithm and Proposed Framework for Task Scheduling in a Grid Environment*. In Proceedings of the 2011 International Conference on Recent Trends in Information Technology, pages 650 – 655, Chennai, Tamil Nadu, 2011.
- [App 2014] *Development Center*. <http://app42paas.shephertz.com/dev-center/>, 2014.
- [Aruldoss *et al.* 2013] M. Aruldoss, T. M. Lakshmi and V. P. Venkatesan. *A Survey*

- on Multi Criteria Decision Making Methods and Its Applications*. American Journal of Information Systems, vol. 1, no. 1, pages 31 – 43, 2013.
- [Bajaj & Agrawal 2004] R. Bajaj and D. P. Agrawal. *Improving Scheduling of Tasks in a Heterogeneous Environment*. IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 2, pages 107 – 118, 2004.
- [Baker et al. 2002] M. Baker, R. Buyya and D. Laforenza. *Grids and Grid Technologies for Wide-area Distributed Computing*. Software Practice Experience, vol. 32, no. 15, pages 1437 – 1466, 2002.
- [Bandyopadhyay et al. 2008] S. Bandyopadhyay, S. Saha, U. Maulik and K. Deb. *A Simulated Annealing Based Multi-objective Optimization Algorithm: AMOSA*. IEEE Transactions on Evolutionary Computation, vol. 12, no. 3, pages 269 – 283, 2008.
- [Becker & Sterling 1995] D. J. Becker and T. Sterling. *BEOWULF A Parallel Workstation for Scientific Computation*. In Proceedings of the International Conference on Parallel Processing, volume 95, pages 11 – 14. CRC Press, Inc., 1995.
- [Bey et al. 2010] K. B. Bey, F. Benhammadia, A. Mokhtarib and Z. Guessoumc. *Independent Task Scheduling in Heterogeneous Environment via Make-span Refinery Approach*. In Proceedings of the International Conference on Machine and Web Intelligence, Algiers, Algiers, pages 211 – 217, 2010.
- [Bit 2014a] *Bittorrent*. <http://www.bittorrent.com/>, 2014.
- [Bit 2014b] *Bitvault*. <http://p2pfoundation.net/Bitvault>, 2014.
- [Blickle & Thiele 1996] Tobias Blickle and Lothar Thiele. *A Comparison of Selection Schemes Used in Evolutionary Algorithms*. Evol. Comput., vol. 4, no. 4, pages 361 – 394, 1996.
- [Blu 2014] *Cloudservices*. <http://www.bluelock.com/>, 2014.

- [Braun *et al.* 2001] T. D. Braun, H. J. Siegel and N. Beck. *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, pages 810 – 837, 2001.
- [Braun *et al.* 2002] T. D. Braun, H. J. Siegel and A. A. Maciejewski. *Static Mapping Heuristics for Tasks with Dependencies, Priorities, Deadlines, and Multiple Versions in Heterogeneous Environments*. In *Proceedings of the International Parallel and Distributed Processing Symposium*, Fort Lauderdale, pages 1 – 8, 2002.
- [Braun *et al.* 2008] T. D. Braun, H. J. Siegel and A. Maciejewski. *Static Mapping Heuristics for Tasks with Dependencies, Priorities, Deadlines, and Multiple Versions In Heterogeneous Environments*. *Journal of Parallel Distributed Computing*, vol. 68, no. 11, pages 1504 – 1516, 2008.
- [Buyya *et al.* 2002] R. Buyya, D. Abramson, J. Giddy and H. Stockinger. *Economic Models for Resource Management and Scheduling in Grid Computing*. *Concurrency and Computation: Practice and Experience (CCPE)*, Wiley Press, vol. 14, no. 13 - 15, pages 1507 – 1542, 2002.
- [Cab 2014] *Clinical Researchers*. <https://cabig.nci.nih.gov/>, 2014.
- [Casanova *et al.* 2000] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environment*. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349 – 363, Cancun, Mexico, 2000.
- [Chapin *et al.* 1999] S. J. Chapin, J. Karpovich and A. Grimshaw. *The Legion Resource Management System*. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 162 – 178, San Juan, Puerto Rico, 1999.
- [Chen & Zhang 2009] W. Chen and J. Zhang. *An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements*. *IEEE*

- Transactions on Systems Man and Cybernetics - Part C: Applications and Reviews, vol. 39, no. 1, pages 29 – 43, 2009.
- [Chervenak *et al.* 2000] A. Chervenak, I. Foster and C. Kesselman. *The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets*. Journal of Network and Computer Applications, vol. 23, no. 3, pages 187 – 200, 2000.
- [Chitra *et al.* 2011] P. Chitra, P. Venkatesh and R. Rajaram. *Application and Comparison of Hybrid Evolutionary Multiobjective Optimization Algorithms for Solving Task Scheduling Problem on Heterogeneous Systems*. Applied Soft Computing, vol. 11, pages 2725 – 2734, 2011.
- [Chou & Abraham 1982] T. C. K. Chou and J. A. Abraham. *Load Balancing in Distributed System*. IEEE Transactions on Software Engineering, vol. 8, pages 401 – 412, 1982.
- [Clo 2014a] *Cloud Computing*. <http://www.dolcera.com/wiki/index.php>, 2014.
- [Clo 2014b] *Cloud Scaling*. <http://www.cloudscaling.com/>, 2014.
- [Clu 2015] *Cluster Computing*. [https://en.wikipedia.org/wiki/Computer\\_cluster](https://en.wikipedia.org/wiki/Computer_cluster), 2015.
- [Darbha & Agrawal 2008] S. Darbha and D. P. Agrawal. *Optimal Scheduling Algorithm for Distributed Memory Machines*. IEEE Transaction on Parallel and Distributed Systems, vol. 9, pages 87 – 95, 2008.
- [Dat 2014] *Datapipe*. <http://www.datapipe.com/>, 2014.
- [Deb *et al.* 2000] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. *A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation, vol. 6, pages 182 – 197, 2000.
- [Deb 2007] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley, New York, 2007.

- [Deelman *et al.* 2005] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz. *Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems*. Scientific Programming 13, IOS Press, pages 219 – 237, 2005.
- [Dong & Akl 2006] F. Dong and S. G. Akl. *Technical Report No. 2006-504 Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*, 2006.
- [Dorigo *et al.* 1996] M. Dorigo, V. Maniezzo and A. Colorni. *The Ant System: Optimization by a Colony of Cooperating Agents*. IEEE Transactions on Systems, Man and Cybernetics - Part B, vol. 26, no. 1, pages 29 – 41, 1996.
- [Durillo & Nebro 2011] J. J. Durillo and A. J. Nebro. *jMetal: A Java Framework for Multi-objective Optimization, Advances in Engineering Software*. Elsevier Science Ltd. Oxford, UK, vol. 42, no. 10, pages 760 – 771, 2011.
- [Durillo *et al.* 2009] J. J. Durillo, A. J. Nebro, F. Luna and E. Alba. *On the Effect of the Steady-State Selection Scheme in Multi-Objective Genetic Algorithms*. In Proceedings of the 5th International Conference Evolutionary Multi-Criterion Optimization, Nantes, France, pages 183 – 197, 2009.
- [Eng 2014] *Deploy, Monitor and Scale Your Application*. <https://www.engineyard.com/>, 2014.
- [Fahringer *et al.* 2005] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Junior and H. Truong. *ASKALON: A Tool Set for Cluster and Grid Computing*. Journal Concurrency and Computation: Practice & Experience- Grid Performance, vol. 17, no. 2 - 4, pages 143 – 169, 2005.
- [Feitelson *et al.* 2004] D. Feitelson, L. Rudolph and U. Schwiegelshohn. *Parallel Job Scheduling - A Status Report*. In Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, New-York, NY, pages 10 – 22, 2004.

- [Foster & Iamnitchi 2003] I. Foster and A. Iamnitchi. *On Death, Taxes, and The Convergence of Peer-to-peer and Grid Computing*. Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, vol. 2735, pages 118 – 128, 2003.
- [Foster & Kesselman 1997] I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of High Performance Computing Applications, vol. 11, pages 115 – 130, 1997.
- [Garg & Singh 2011] R. Garg and A. K. Singh. *Multi-Objective Optimization to Workflow Grid Scheduling using Reference Point based Evolutionary Algorithm*. International Journal of Computer Applications, vol. 22, no. 6, pages 44 – 49, 2011.
- [Garg *et al.* 2008] S. K. Garg, P. Konugurthi and R. Buyya. *A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids*. In Proceedings of the 16th International Conference on Advanced Computing and Communication (ADCOM 2008), Chennai, India, pages 1 – 9, 2008.
- [Garg *et al.* 2010] S. K. Garg, R. Buyya and H. J. Siegel. *Cost Trade-Off Management for Scheduling Parallel Applications on Utility Grids*. Future Generation Computer Systems, vol. 26, pages 1344 – 1355, 2010.
- [Golconda *et al.* 2004] K. S. Golconda, F. Ozguner and A. Dogan. *A Comparison of Static QoS-Based Scheduling Heuristics for a Meta-Task with Multiple QoS Dimensions in Heterogeneous Computing*. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, USA, pages 1 – 14, 2004.
- [Goldberg 1989] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.
- [Gong *et al.* 2002] L. G. Gong, X. H. Sun and E. F. Watson. *Performance Modeling and Prediction of Non-dedicated Network Computing*. IEEE Transactions on Computers, vol. 51, no. 9, pages 1041 – 1055, 2002.

- [Gri 2005] *Grid Service Broker*. <http://www.cloudbus.org/broker/>, 2005.
- [Gri 2014] *Grid Simulator*. <http://www.buyya.com/gridsim/>, 2014.
- [Gri 2015] *Grid Computing*. <http://computer.howstuffworks.com/grid-computing.htm>, 2015.
- [He *et al.* 2003] X. S. He, X. H. Sun and G. V. Laszewski. *QoS Guided Min- $\lambda$ min Heuristic for Grid Task Scheduling*. *Journal Computer Science Technology*, vol. 18, no. 4, pages 442 – 451, 2003.
- [Her 2014] *Build, Run, and Scale Apps*. <https://www.heroku.com/>, 2014.
- [Hoscheck *et al.* 2000] W. Hoscheck, F. J. Jaen-Martinez, A. Samar, H. Stockinger and K. Stockinger. *Data Management in an International Data Grid Project*. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 77 – 90, 2000.
- [Hoschek *et al.* 2000] W. Hoschek, J. Jaen-Martinez and A. Samar. *Data Management in an International Data Grid Project*. In *Proceedings of the first International Workshop on Grid Computing*, pages 1 – 15, Banaglore, India, 2000.
- [Huang *et al.* 2007] K. C. Huang, P. C. Shih and Y. C. Chung. *Towards Feasible and Effective Load Sharing in a Heterogeneous Computational Grid*. In *Proceedings of the Second International Conference on Grid and Pervasive Computing*, volume 4459, pages 229 – 240, 2007.
- [Huang *et al.* 2009] K. Huang, C. Shih and Y. Chung. *Adaptive Processor Allocation for Moldable Jobs in Computational Grid*. *International Journal of Grid and High Performance Computing*, vol. 1, no. 1, pages 10 – 21, 2009.
- [Ishibuchi & Murata 1998] Hisao Ishibuchi and Tadahiko Murata. *A multi-objective genetic local search algorithm and its application to flowshop scheduling*. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 28, no. 3, pages 392 – 403, 1998.

- [Jackson *et al.* 2001] D. Jackson, Q. Snell and M. Clement. *Core Algorithms of The Maui Scheduler*. In Proceedings of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, pages 87 – 102, 2001.
- [Joshy & Craig 2003] J. Joshy and F. Craig. Grid computing. International Business Machines Press, 2003.
- [Kamalam & Muralibhaskaran 2010] G. K. Kamalam and V. Muralibhaskaran. *A New Heuristic Approach: Min-mean Algorithm for Scheduling Meta-Tasks on Heterogeneous Computing Systems*. International Journal of Computer Science and Network Security, vol. 10, no. 1, pages 24 – 31, 2010.
- [Kang & Agrawal 2000] O. Kang and D. P. Agrawal. *S3MP: A Task Duplication Based Scalable Scheduling Algorithm for Symmetric Multiprocessors*. In Proceedings of the 14th International Parallel and Distributed Processing Symposium, pages 451 – 456, 2000.
- [Khalifa *et al.* 2007] A. S. Khalifa, R. A. Ammar, T. A. Fegrany and M. E. Khalifa. *A Preemptive Version of the Min-min Heuristic for Dynamically Mapping Meta Task on a Distributed Heterogeneous Environment*. In Proceedings of the IEEE International Symposium on Signal Proceeding and Information Technology, Giza, pages 537 – 542, 2007.
- [Kirkpatrick *et al.* 1983] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. *Optimization by Simulated Annealing*. Journal of Science, vol. 220, no. 4598, pages 671 – 680, 1983.
- [Konak *et al.* 2006] A. Konak, D. W. Coit and A. E. Smith. *Multi-objective Optimization Using Genetic Algorithms: A Tutorial*. Reliability Engineering & System Safety, vol. 91, no. 9, pages 992 – 1007, 2006.
- [Krueger & Finkel 1984] P. Krueger and V. Finkel. *An Adaptive Load Balancing Algorithm for a Multicomputer*. Technical Report No. 539, April 1984, pages 1 – 21, 1984.



- [Kumar *et al.* 2009] A. Kumar, N. Chaubey and S. Yakkali. *Immediate Mode Scheduling Methods for Independent Jobs on Open Online Heterogeneous Systems*. In Proceedings of the 15th International Conference on High Performance Computing, pages 12 – 17, Banaglore, India, 2009.
- [Kureger & Livny 1987] P. Kureger and M. Livny. *The Diverse Objectives of Distributed Scheduling Policies*. In Proceedings of the 7th IEEE International Conference on Distributed Computing Systems, Berlin, Germany, pages 242 – 249, 1987.
- [Kurowski *et al.* 2007] K. Kurowski, J. Nabrzysk, A. Oleksiak and J. Weglarz. *GSSIM - Grid Scheduling Simulator*. Computational Methods in Science and Technology, vol. 13, no. 2, pages 121 – 129, 2007.
- [Lau *et al.* 2006] S. Lau, Q. Lu and K. Leung. *Adaptive Load Distribution Algorithms for Heterogeneous Distributed Systems with Multiple Task Classes*. Journal of Parallel and Distributed Computing, vol. 66, no. 2, pages 163 – 180, 2006.
- [Lee & Zomaya 2006] Y. C. Lee and A. Y. Zomaya. *A Grid Scheduling Algorithm for Bag-of-Tasks Applications Using Multiple Queues with Duplication*. In Proceedings of the 5th IEEE International Conference on Computer and Information Science, pages 5 – 10, 2006.
- [Lee *et al.* 2006] L. T. Lee, C. Liang and H. Chang. *An Adaptive Task Scheduling System for Grid Computing*. In Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Seoul, pages 1 – 6, 2006.
- [Legrand *et al.* 2003] A. Legrand, L. Marchal and H. Casanova. *Scheduling Distributed Applications: The SimGrid Simulation Framework*. In Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, page 138 – 145, 2003.
- [Lhc 2014] *Large Hadron Collider*. <http://home.web.cern.ch/topics/large-hadron-collider>, 2014.

- [Li *et al.* 2010] A. Li, N. Yao and P. Hong. *A Cost and Time Balancing Algorithm for Scheduling Parallel Tasks on Computing Grid*. In Proceedings of the International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE), pages 185 – 188, Hong Kong, 2010.
- [Lifka 1995] D. A. Lifka. *The ANL/IBM SP Scheduling System*. In Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA, pages 295 – 303, 1995.
- [Loa 2014] *What is Load Leveler Resource Manager*. <http://www-01.ibm.com/support/knowledgecenter/>, 2014.
- [Lublin & Feitelson 2003] U. Lublin and D. Feitelson. *The Workload on Parallel Supercomputers: Modeling The Characteristics of Rigid Jobs*. Journal of Parallel and Distributed Computing, vol. 63, pages 1105 – 1122, 2003.
- [Menascé & Almeida 1990] D. Menascé and V. Almeida. *Cost-performance Analysis of Heterogeneity in Supercomputer Architectures*. In Proceedings of the 1990 ACM/IEEE Conference on Supercomputing, New York, New York, USA, pages 169 – 177, 1990.
- [Muallem & Feitelson 2001] A. W. Muallem and D. G. Feitelson. *Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling*. IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 6, pages 529 – 543, 2001.
- [Munir *et al.* 2008] E. U. Munir, J. Li, S. Shi, Z. Zou and Q. Rasool. *A New Heuristic for Task Scheduling in Heterogeneous Computing Environment*. Journal of Zhejiang University Science, vol. 9, pages 1715 – 1723, 2008.
- [Nahrstedt *et al.* 1998] K. Nahrstedt, H. Chu and S. Narayan. *QoS-aware Resource Management for Distributed Multimedia*. Journal on High-Speed Networking - Special Issue on Multimedia Networking, vol. 7, no. 3 - 4, pages 229 – 257, 1998.

- [Nebro *et al.* 2008] A. J. Nebro, J. J. Durillo, C. A. Coello, F. Luna and E. Alba. *A Study of Convergence Speed in Multi-Objective Metaheuristics*. In Proceedings of the 10th International Conference Parallel Problem Solving from Nature, Dortmund, Germany, pages 763 – 772, 2008.
- [Nir 2014] *Nirmod14-G*. <http://www.gridbus.org/>, 2014.
- [Noraini & G. 2011] M. R. Noraini and John G. *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*. In Proceedings of the 2011 International Conference of Computational Intelligence and Intelligent Systems, London, United Kingdom, pages 1134 – 1139, 2011.
- [Ope 2014] *Develop, Host and Scale APPS in the Cloud*. <https://www.openshift.com/>, 2014.
- [Ousterhout 1982] J. K. Ousterhout. *Scheduling Techniques for Concurrent Systems*. In Proceedings of the Third International Conference on Distributed Computing Systems, volume 18, pages 22 – 30, 1982.
- [Pandey *et al.* 2010] S. Pandey, L. Wu, S. M. Guru and R. Buyya. *A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments*. In Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications, pages 400 – 407, 2010.
- [Parsa & Entezari-Malekir 2009] S. Parsa and R. Entezari-Malekir. *RASA: A New Grid Task Scheduling Algorithm*. International Journal of Digital Content Technology and its Applications, vol. 3, no. 4, pages 91 – 99, 2009.
- [Pee 2015] *Peer-to-peer Computing*. <https://en.wikipedia.org/wiki/Peer-to-peer>, 2015.
- [Pla 2005] *IBM Platform Computing*. <http://www-03.ibm.com/systems/platformcomputing/>, 2005.

- [Prodan & Fahringer 2005] R. Prodan and T. Fahringer. *Dynamic Scheduling of Scientific Workflow Applications on the Grid: A Case Study*. In Proceedings of the 20th Symposium of Applied Computing, ACM Press, pages 687 – 694, 2005.
- [Radulescu & Van Gemund 1999] A. Radulescu and A. J. C. Van Gemund. *On the Complexity of List Scheduling Algorithms for Distributed Memory Systems*. In Proceedings of the 13th International Conference on Supercomputing, Portland, Oregon, USA, November, pages 68 – 75, 1999.
- [Ranaweera & Agrawal 2000] S. Ranaweera and D. P. Agrawal. *A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems*. In Proceedings of the 14th International Parallel and Distributed Processing Symposium , Cancun, Mexico, May, pages 445 – 450, 2000.
- [Sahu & Chaturvedi 2011] R. Sahu and A. K. Chaturvedi. *Many-Objective Comparison of Twelve Grid Scheduling Heuristics*. International Journal of Computer Applications, vol. 13, no. 6, pages 9 – 17, 2011.
- [Sakellariou & Zhao 2004] R. Sakellariou and H. Zhao. *A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems*. Scientific Programming, vol. 12, no. 4, pages 253 – 262, 2004.
- [Sample *et al.* 2002] N. Sample, P. Keyani and G. Wiederhold. *Scheduling Under Uncertainty: Planning for the Ubiquitous Grid*. In Proceedings of the 5th International Conference on Coordination Models and Languages, Lecture Notes in Computer Science, pages 300 – 316. Springer-Verlag, 2002.
- [Schaffer 1985] J. D. Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. In Proceedings of the 1st International Conference on Genetic Algorithms, pages 93 – 100, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc.
- [Schwiegelshohn & Yahyapour 1998] U. Schwiegelshohn and R. Yahyapour. *Analysis of First-come First-serve Parallel Job Scheduling*. In Proceedings of the

- Ninth Annual ACM-SIAM symposium on Discrete algorithms, pages 1 – 10, 1998.
- [Sen & Yang 1998] P. Sen and J. B. Yang. Multiple criteria decision support in engineering design. Springer-Verlag Berlin Heidelberg, New-york, 1998.
- [Set 2014] *Seti*. <http://setiathome.ssl.berkeley.edu/>, 2014.
- [Seymour *et al.* 2005] K. Seymour, A. YarKhan, S. Agrawal and J. Dongarra. *Net-Solve: Grid Enabling Scientific Computing Environments*. Grid Computing and New Frontiers of High Performance Processing, Advances in Parallel Computing, Elsevier, vol. 14, pages 33 – 51, 2005.
- [Sfiligoi 2008] I. Sfiligoi. *glideinWMS - A Generic Pilot-based Workload Management System*. In Proceedings of the International Conference on Computing in High Energy and Nuclear Physics, New York, USA, pages 1 – 9, 2008.
- [Shih *et al.* 2013] P. Shih, K. Huang, C. Lee, I. Chung and Y. Chung. *Temporal Look-ahead Processor Allocation Method for Heterogeneous Multi-cluster Systems*. Journal Parallel Distributed Computing, vol. 73, no. 12, pages 1661 – 1672, 2013.
- [Shivaratri *et al.* 1992] N. G. Shivaratri, P. Krueger and M. Singhal. *Load Distributing in Locally Distributed System*. IEEE Transaction on Computer, vol. 25, no. 12, pages 33 – 44, 1992.
- [Shmueli & Feitelson 2005a] E. Shmueli and D. G. Feitelson. *Backfilling with Lookahead to Optimize the Packing of Parallel Jobs*. Journal of Parallel and Distributed Computing, vol. 65, no. 9, pages 1090 – 1107, 2005.
- [Shmueli & Feitelson 2005b] E. Shmueli and D. G. Feitelson. *Backfilling with Lookahead to Optimize the Packing of Parallel Jobs*. Journal of Parallel Distributed Computing, vol. 65, no. 9, pages 1090 – 1107, 2005.
- [Sim 2014] *SimGrid Portable*. <http://simgrid.gforge.inria.fr/>, 2014.

- [Singhal & Niranjana 2006] M. Singhal and G. S. Niranjana. Advance concepts in operating system. Tata McGraw-Hill Thirteenth Edition, 2006.
- [Singhal & Shivaratri 1998] M. Singhal and N. G. Shivaratri. Advanced concepts in operating systems. McGraw-Hill, ISBN 0-07-13668-1, 1998.
- [Sky 2014] *Skype*. <http://www.skype.com/en/>, 2014.
- [Sun 2014] *Sun Cluster*. <http://gridengine.sunsource.net/>, 2014.
- [Takefusa *et al.* 1999] A. Takefusa, S. Matsuoka and H. Nakada. *Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms*. In Proceedings of the Eighth International Symposium on High Performance Distributed Computing Proceedings, pages 97 – 104. IEEE Computer Society, 1999.
- [Tanenbaum & Bos 2015] A. S. Tanenbaum and H. Bos. Modern operating systems. Pearson Higher Education, United State, 2015.
- [Thain *et al.* 2005] D. Thain, T. Tannenbaum and M. Livny. *Distributed Computing in Practice: The Condor Experience*. Journal Concurrency and Computation: Practice and Experience, vol. 17, pages 323 – 356, 2005.
- [Topcuouglu *et al.* 2002] H. Topcuouglu, S. Hariri and M. Wu. *Performance-effective and Low-complexity Task Scheduling for Heterogeneous Computing*. IEEE Transaction on Parallel and Distributed Systems, vol. 13, no. 4, pages 260 – 274, 2002.
- [Wieczorek *et al.* 2008] M. Wieczorek, S. Podlipnig, R. Prodan and T. Fahringer. *Bi-Criteria Scheduling of Scientific Workflows for the Grid*. In Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, pages 9 – 16, 2008.
- [Win 2014] *Windows Cluster*. <http://technet.microsoft.com/en-us/library/>, 2014.
- [Wolski *et al.* 1999] R. Wolski, N. T. Spring and J. Hayes. *Network Weather Service: A Distributed Resource Performance Forecasting Service for Meta Computing*.

- Journal of Future Generation Computing System, vol. 15, no. 6, pages 757 – 768, 1999.
- [Wu *et al.* 2000] M. Wu, W. Shu and H. Zhang. *Segmented Min-min: A Static Mapping Algorithm for Meta-tasks on Heterogeneous Computing Systems*. In Proceedings of the 9th Heterogeneous Computing Workshop, Cancun, Mexico, pages 375 – 385, 2000.
- [Xhafa & Abraham 2010] F. Xhafa and A. Abraham. *Computational Models and Heuristic Methods for Grid Scheduling Problems*. Future Generation Computer System, vol. 26, no. 4, pages 608 – 621, 2010.
- [Yan & Chapman 2005] Y. Yan and B. Chapman. *Comparative Study of Distributed Resource Management Systems*. In Proceedings of the 2005 American Control Conference, volume 2, pages 1484 – 1490, 2005.
- [Yarkhan & Dongarra 2002] A. Yarkhan and J. J. Dongarra. *Experiments with Scheduling Using Simulated Annealing in a Grid Environment*. Lecture Notes in Computer Science, pages 232 – 242, 2002.
- [Yu 2007] J. Yu. *QoS-based Scheduling of Workflows on Global Grids*. PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia., 2007.
- [Zhao *et al.* 2006] Y. Zhao, M. Wilde and I. Foster. *Virtual Data Grid Middleware Services for Data-Intensive Science*. Concurrency and Computation: Practice and Experience, vol. 18, no. 6, pages 595 – 608, 2006.
- [Zheng *et al.* 2006] S. Zheng, W. Shu and L. Gao. *Task Scheduling using Parallel Genetic Simulated Annealing Algorithm*. In Proceedings of the IEEE International Conference Service Operations Logist, pages 46 – 50, 2006.
- [Zitzler & Thiele 1999] E. Zitzler and L. Thiele. *Multiobjective Evolution-ary Algorithms: A Comparative Case Study and the Strength Pareto Approach*. IEEE

Transactions on Evolutionary Computation, vol. 3, no. 4, pages 257 – 271, 1999.

[Zomaya & Teh 2001] A. Y. Zomaya and Y. Teh. *Observations on Using Genetic Algorithms for Dynamic Load-Balancing*. IEEE Transactions on Parallel Distributed Systems, vol. 12, no. 9, pages 899 – 911, 2001.



# Publications

## Conference Papers

1. Sunita Bansal, and Chittaranjan Hota, Goodwill Based Scheduling Algorithm for Economy Grid, Third IEEE International Advance Computing Conference, Ajay Kumar Garg Engineering College (AKGEC), Ghaziabad, India, pages 55-60, Published in IEEE Xplore, February, 2013.
2. Sunita Bansal, and Chittaranjan Hota, Efficient Algorithm on Heterogeneous Computing System, Third International Conference in Recent Trend in Information Systems, ReTIS'11, Javadpur University, Kolkata, India, pages 57-61, Published in IEEE Xplore, December, 2011.
3. Sunita Bansal, Gowtham K, and Chittaranjan Hota, Novel Adaptive Scheduling Algorithm for Computational Grid, Third International Conference on Internet Multimedia Services Architecture and Applications (IEEE IMSAA-IWAP2PT ), International Institute of Information Technology, Bangalore, India, pages 158-162, Published in IEEE Xplore, December 2009.
4. Sunita Bansal, and Chittaranjan Hota, Priority-based Job Scheduling in Distributed Systems, Third International Conference on Information Systems, Technology and Management, Information Systems and Technology Management, Ghaziabad, India, volume 31, pages 110-118, Published in Springer-Verlag Berlin Heidelberg, March 2009.

## Journal Papers

1. Sunita Bansal, and Chittaranjan Hota, Hybrid Multi-objective Workflow Scheduling on Economic Grids, INFOCOMP Journal of Computer Science, Brazil, volume 13, number 1, June 2014, pages 12-20.

2. Sunita Bansal, and Chittaranjan Hota, Distributed Scheduling on Utility Grids, Romanian Journal of Information Science and Technology (SCI Expanded), volume 16, number 4, 2013, pages 373-392.
3. Sunita Bansal, Bhavik Kothari, and Chittaranjan Hota, Dynamic Task-Scheduling in Grid Computing using Prioritized Round Robin Algorithm, International Journal of Computer Science Issues (IJCSI), volume 8, issue 2, 2011, pages 472-477.
4. Sunita Bansal, and Chittaranjan Hota, Efficient Refinery Scheduling Heuristic in Heterogeneous Computing Systems, Journal of Advances in Information Technology, volume 2, number 3 (Special Issue: Advanced Algorithms), 2011, pages 159-164, Academy Publisher Finland.
5. Sunita Bansal, Divya Gupta, and Chittaranjan Hota, Adaptive Decentralized Load Sharing Algorithms with Multiple Job Transfers in Distributed Computing Environments, International Joint Journal Conference in Computer, Electronics and Electrical, CEE 2009, International Journal of Recent Trends in Engineering volume 2, number 2, 2009, pages 217-221, Academy Publisher Finland.

# Biographies

## **Brief Biography of the Candidate**

Sunita Bansal is working in Birla Institute of Technology and Science, Pilani since the year 2005. She received her M Tech. (CS) and M. Sc. (CS) degree from Banasthali Vidyapith, Banasthali, Rajasthan, India in 2005, 2003 respectively. She is member of Computer Society of India (CSI); Indian Society for Technical Education (ISTE); Indian Science Congress Association (ISCA); International Association of Engineers, USA; and International Association of Computer Science and Information Technology, Singapore. Her research interest is scheduling in distributed and parallel systems.

## **Brief Biography of the Supervisor**

Chittaranjan Hota is currently Full Professor of Computer Science & Engineering and the Associate Dean of Admissions at Birla Institute of Technology and Science-Pilani, Hyderabad Campus, Hyderabad. He served as the founding Head in the Dept. of Computer Science and Engineering at BITS, Hyderabad. He has a Ph.D. in Networks and Information Security from the Dept. of Computer Science and Engineering, BITS, Pilani. Prior to this he has a Masters in Engineering (Computer Sc. & Engineering) from Thapar with First class (honors), and a Bachelors in Engineering (Computer Engineering) from Amravati (MS) with First class. He has been at BITS-Pilani since past fifteen years, and overall since past twenty-five years at Indian universities. He has been a visiting researcher and visiting professor over two months to a semester at University of New South Wales, Sydney; University of Cagliari, Italy; Aalto University, Finland; and City University, London in the past. His research has been funded by University Grants Commission, New Delhi; Department of Information Technology, New Delhi; and Tata

Consultancy Services, India. He has guided three Ph.D. students and currently guiding six Ph.D. students in the areas of Wireless networks, Information security, and Healthcare informatics. He is the recipient of Australian Vice Chancellors' Committee award, recipient of Erasmus Mundus fellowship from European commission, Italian ministry of education fellowship, and recipient of Certificate of Excellence from K.R. Faculty Excellence Award at BITS Pilani. He has published extensively in peer-reviewed journals and conferences. He has edited LNCS volumes. He is a member of IEEE, ACM, IE, and ISTE.