

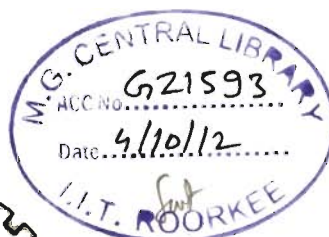
TOWARDS MULTI-OBJECTIVE SCHEDULING STRATEGIES FOR GRID COMPUTING ENVIRONMENTS

A THESIS

*Submitted in partial fulfilment of the
requirements for the award of the degree
of
DOCTOR OF PHILOSOPHY
in
COMPUTER SCIENCE AND ENGINEERING*

by

AMIT AGARWAL



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

OCTOBER, 2011

**©INDIAN INSTITUTE OF TECHNOLOGY ROORKEE, ROORKEE – 2011
ALL RIGHTS RESERVED**



INDIAN INSTITUTE OF TECHNOLOGY ROORKEE ROORKEE

CANDIDATE'S DECLARATION


I hereby certify that the work which is being presented in this thesis entitled **TOWARDS MULTI-OBJECTIVE SCHEDULING STRATEGIES IN GRID COMPUTING ENVIRONMENTS** in partial fulfilment of the requirements for the award of the Degree of DOCTOR OF PHILOSOPHY and submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, is an authentic record of my own work carried out during a period from January, 2007 to October, 2011 under the supervision of Dr. Padam Kumar, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, India.

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

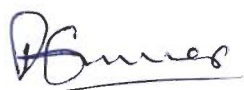

(AMIT AGARWAL)


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

Date: 7-10-2011


(Padam Kumar)
Supervisor

The Ph.D. Viva-Voce Examination of **Mr. Amit Agarwal**, Research Scholar, has been held on 2-1-2012.


Signature of Supervisor


Signature of External Examiner

ACKNOWLEDGEMENTS

This thesis is the culmination of a long journey throughout which I have received support from many valuable people whom I wish to acknowledge here. First and foremost, I feel proud to have Dr. Padam Kumar, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, as my supervisor, who is abundantly helpful and offered invaluable assistance, patience guidance and excellent advice throughout the course of this work. I would like to express my heartfelt gratitude towards him for encouraging me to carry out my research in the area of parallel and distributed computing as well as providing me all the necessary guidance and inspirational support, without which this work would not have been in the present shape.

I also feel obliged to the Research Committee comprising of Prof. Vinod Kumar, Prof. A. K. Sarje and Prof. D. K. Mehra for their critical and constructive suggestions and comments during the initial and final scrutiny of this research work.

I also extend my sincere thanks to Dr. S. N. Sinha, Professor & Head, Department of Electronics and Computer Engineering, for providing facilities and support needed for this work.

I also extend heartily gratitude to Dr. Ankush Mittal (Director Research, COER, Roorkee) for their valuable suggestions for improving my thesis.

I feel indebted to the staff of Institute Computer Centre and Department of Electronics & Computer Engineering for providing desired help as and when required. I have a special word of thanks for Dr. N. K. Gupta, System Programmer at ICC, IIT Roorkee for providing necessary facilities to carry out simulation work as well as building grid infrastructure out of the computational resources of Institute Computer Centre.

I express my deep gratitude to my parents R. K. Agarwal and Reena Agarwal for always posing great confidence and instilling faith in me. I also owe respects to my parents-in-law Suresh Chandra and Rekha Singhal, who always showered their blessing on me.

Finally, I take this opportunity to express my profound gratitude to my beloved wife Shikha and my lovely son Divyansh for their moral support and patience during my study at IIT Roorkee.

Above all, I am grateful to the almighty 'God' for his blessings.


(Amit Agarwal)

ABSTRACT

In recent years, there has been increasing interest in using network-based resources for large scale data-intensive computation problems. The Grid computing systems are rapidly gaining importance due to their capability to deal with the continuously rising demands of computational and storage resources. A grid may consist of resources owned by a number of different organizations, within which sharing arrangements have been established. Grid computing can be defined as coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations.

Scheduling is an important research topic in Grid computing. The purpose of grid is to offer high quality services to members of Virtual Organizations (VOs) based on the efficient use of the available resources. This goal cannot be achieved without a good scheduling strategy applied to the local level (clusters) and global level (entire system) of Grids. Since the grid scheduling problem is NP-hard, we can afford only sub-optimal solutions to this problem. Grid scheduling scenario has additional complexities due to scattered data storages and consequent delays in data transfers, coordination required in the execution of linked tasks and arranging the required data interchange. Further challenges are related to the dynamic nature of Grid systems where the changing availability and quality of resources makes the scheduling decisions still more difficult.

Optimally scheduling a single set of dependent tasks to multiple heterogeneous resources is an algorithmically hard problem. This problem becomes even harder when optimization of multiple criteria (at times conflicting) is required. Quality of Service (QoS) requirements of applications and resources add an extra layer of complexity to scheduling while matching the tasks to resources. Due to the dynamic nature of grid resources, the scheduling of scientific application tasks is still an issue and is an open problem. This

thesis attempts to improve upon these frontages by contributing new single criteria and multi-criteria scheduling heuristics to satisfy performance objectives like makespan, economic cost, resource consumption, etc while meeting the QoS requirements like trustworthiness, availability, network bandwidth, etc.

We have classified the research work in three parts: (1) Single Criteria Scheduling – Scheduling of application tasks meeting a single objective function like makespan or economic cost, (2) Bi-Criteria Scheduling – Scheduling of tasks meeting two objective functions simultaneously, and (3) QoS Oriented Multi Criteria Scheduling – Scheduling of tasks meeting more than one objective functions while respecting QoS requirements like bandwidth, availability and trustworthiness of resources.

The research work in *single criteria scheduling* has been classified in two categories: (1) Time based scheduling, and (2) Cost based scheduling. In *time based scheduling*, we have developed a new scheduling heuristic called *Scheduling with Heterogeneity using Critical Path* (SHCP) algorithm for grid computing systems. The performance of workflow scheduling greatly depends on task sequence obtained using b-level (longest directed path considering average computation and communication cost from concerned task to exit task). But, in grids, the above approach does not fit due to high heterogeneity in computation costs of tasks on different resources and communication costs of data transfers among different tasks (scheduled on different resources) due to variable network bandwidth. In this research, computation and communication heterogeneity factors have been formulated based on 'Standard Deviation' which reflects the spread of execution time/ communication time of a task/ data transfer in workflow. The expected computation cost of each task is determined using computation heterogeneity factor and the expected communication cost is computed using communication heterogeneity factor rather than taking average computation and communication costs. The priority based task sequence is generated using these costs. The tasks on critical path in workflow are given

priority in the task sequence. The SHCP algorithm is applied to this task sequence to generate the schedules. The simulated result analysis shows that SHCP generates comparatively shorter schedules for large workflow applications in grids.

In *cost based scheduling*, task duplication and compaction strategy has been adopted in scheduling. This strategy adopts a mechanism to minimize the number of duplications (i.e., duplication cost overhead) to optimize the schedules. The scheduling algorithms (HED and RD) have been developed and validated for heterogeneous and homogeneous computing systems respectively. In this approach, the schedule is obtained using duplication based scheduling and then useless duplications and unproductive sub-schedules are removed from the above schedule to minimize the effective consumption of resources. The simulated result analysis shows that HED (Heterogeneous Economical Duplication) and RD (Reduced Duplication) algorithms generate comparable schedules with remarkably less duplication overheads and less processor consumption in heterogeneous and homogeneous computing environments respectively. The EDS-G (Economical Duplication Based Scheduling in Grids) inspired from HED strategy has been developed and validated in a simulated grid environment.

In *bi-criteria scheduling*, two heuristics have been suggested i.e., *compaction based bi-criteria scheduling* (SODA) and *two phase bi-criteria scheduling* (DBSA). A compaction based scheduling (SODA) comprises two stages: (1) duplication based scheduling – optimizes the primary criterion, i.e., execution time, (2) compaction of schedules – minimizes the processor requirements and optimizes secondary criterion, i.e., economic cost without increasing the makespan obtained in primary scheduling. It exploits duplication strategy to obtain a shorter schedule in primary stage (primary scheduling) in order to minimize makespan (primary criteria). In secondary scheduling, the primary schedule is scanned to identify and remove the useless duplications

and unproductive schedules, if any, without letting it to increase the makespan obtained in primary scheduling. In addition, the above modified schedule is further improved by swapping tasks among resources (from costlier to cheaper resource) in order to minimize economic cost (secondary criteria) without affecting the makespan. The experimental results reveal that the proposed approach generates schedules with low processor requirements which are fairly optimized for both economic cost and makespan for executing DAG applications in the grid environments.

The Two phase bi-criteria scheduling (DBSA) is an extension of SODA approach which adopts the concept of sliding constraint in secondary scheduling phase. The primary phase of this approach is similar to SODA algorithm. In next phase, the primary schedule is first modified in order to minimize useless duplications and unproductive schedules, then it is improved by swapping tasks among resources (from costlier to cheaper resource) in order to minimize economic cost (secondary criteria) while letting the makespan to deteriorate within sliding constraint limits. This algorithm (DBSA) is developed and validated in the simulated grid environments. The simulated results show that DBSA surpasses SODA heuristic in terms of economic cost due to sliding makespan. It also generates better schedules which are fairly optimized in respect of both makespan and economic cost.

Further, *QoS oriented multi-criteria scheduling* is discussed for (a) Trust based and (b) Availability aware scheduling. The research presented here considers the QoS constraints (trust, availability) of grid resources for scheduling application tasks using multiple criteria scheduling approach. This research work has been classified as trust based multi-criteria scheduling and availability aware QoS scheduling. In *trust based multi-criteria scheduling*, trust has been used as QoS requirement of workflow tasks. Trust is a major concern of resource users and owners in the grid environment with uncountable and at times unreliable nodes. An intelligent scheduling mechanism is essential which

may require several different criteria to be considered simultaneously when evaluating the quality of solution or a schedule. A trust-oriented multi-objective scheduling (Trust-MOS) heuristic is designed to schedule the tasks to highly trustworthy resources in the grid environment. This approach has been divided into two phases. In primary phase, each task is scheduled on to a resource which fulfills the trust requirements and minimizes the makespan. The schedule generated in primary phase is then modified using a greedy approach to minimize the economic cost as well keeping the makespan within specified limit based on a sliding constraint. The Trust-MOS scheduling algorithm has been implemented in simulated grid environments. The simulated results reveal that Trust-MOS generates good quality schedules in terms of trust overheads, and reduces probabilities of task failure, hence is well suited for executing tasks in a secure manner.

Availability aware QoS scheduling strategy considers availability as QoS of compute resource and bandwidth as QoS of underlying network in grid. A novel availability aware QoS oriented algorithm (AQuA) is developed for executing applications in grid environment. The application tasks specify the availability and bandwidth as QoS requirements to select the highly available compute resource over dedicated network. Such tasks get priority in scheduling over the qualified resources. After mapping of such tasks, the other tasks not having QoS requirements are mapped. The AQuA algorithm has been validated in simulated grid environment and results have been compared with the exiting QoS Guided Min-Min (QGMM) algorithm. The comparative result analysis shows that AQuA is able to prioritize the highly available grid resources and therefore increases the reliability to successfully execute applications without adversely affecting the makespan. In future, the research has been proposed to be extended towards the development of multi-criteria scheduling techniques for simultaneous execution of multiple workflows in grids.

CONTENTS

CANDIDATE’S DECLARATION	i
ACKNOWLEDGEMENTS	ii
ABSTRACT	iv
CONTENTS	ix
LIST OF FIGURES	xiv
LIST OF TABLES	xviii
CHAPTER 1: Introduction	1
1.1 Overview.....	1
1.2 Grid Computing.....	2
1.2.1 Grid Architecture.....	4
1.2.2 Grid Computing in Research.....	7
1.2.3 Grid Computing in Business.....	8
1.3 Grid Scheduling and Research Issues.....	11
1.4 Motivations and Objectives.....	16
1.5 Organization of Thesis.....	19

CHAPTER 2: Backgrounds and Literature Review.....	20
2.1 Review of Scheduling Strategies.....	20
2.1.1 Independent Task Scheduling.....	22
2.1.2 Workflow Scheduling.....	27
2.1.3 Dynamism of Grid.....	36
2.1.4 Dynamic Rescheduling.....	39
2.1.5 Nature's Laws Inspired Scheduling.....	41
2.2 Research Directions.....	43
CHAPTER 3: Single Criterion Scheduling.....	45
3.1 Overview.....	45
3.2 Time Based Scheduling.....	46
3.2.1 Preamble.....	46
3.2.2 Grid Resource Model.....	48
3.2.3 Workflow Application Model.....	49
3.2.4 Matter of Heterogeneity in Grids.....	49
3.2.5 Heterogeneity Aware Critical Path Based Scheduling.....	51
3.2.6 Performance Comparisons and Result Analysis.....	54
3.3 Cost Based Scheduling.....	57

3.3.1	Duplication Cost Based Scheduling.....	57
3.3.2	Resource Model.....	59
3.3.3	Reducing Duplications – Needs and Approaches	62
3.3.4	Performance Comparisons and Result Analysis	67
3.4	Summary and Discussions.....	72
 CHAPTER 4: Bi-Criteria Scheduling.....		73
4.1	Overview.....	73
4.2	Compaction Based Bi-Criteria Scheduling	75
4.2.1	Preamble.....	75
4.2.2	Grid Resource Model.....	78
4.2.3	Workflow Application Model.....	79
4.2.4	Performance Metrics.....	80
4.2.5	SODA Algorithm.....	82
4.2.6	Performance Comparisons and Result Analysis.....	87
4.3	Two Phase Bi-Criteria Scheduling	90
4.3.1	Preamble.....	90
4.3.2	Grid Resource Model.....	93
4.3.3	Workflow Application Model.....	93

4.3.4	DBSA Algorithm.....	93
4.3.5	Performance Comparisons and Result Analysis	97
4.4	Summary and Discussions.....	101
 CHAPTER 5: QoS Oriented Multiple Criteria Scheduling.....		102
5.1	Overview.....	102
5.2	Trust Oriented Multi Criteria Scheduling.....	103
5.2.1	Preamble.....	103
5.2.2	Trust Model for Grids.....	106
5.2.3	Performance Metrics.....	109
5.2.4	Trust–MOS Scheduling Algorithm.....	111
5.2.5	Performance Comparisons and Result Analysis	116
5.3	Availability Aware QoS Oriented Scheduling.....	119
5.3.1	Preamble.....	120
5.3.2	Related Work.....	121
5.3.3	Grid Resource Model.....	123
5.3.4	Grid Application Model.....	125
5.3.5	AQuA Scheduling Algorithm.....	126
5.3.6	Simulation and Result Analysis.....	131

5.4	Summary and Discussions.....	136
CHAPTER 6: Conclusions and Future Directions.....		138
6.1	Conclusions.....	138
6.2.	Future Research Directions.....	140
BIBLIOGRAPHY		141
VITAE		160
PUBLICATIONS		161
APPENDIX – A: Simulation Using MATLAB		163
APPENDIX – B: Stepwise Trace of DBSA Algorithm		168
APPENDIX – C: MATLAB Code of AQUA Algorithm		171

LIST OF FIGURES

Figure 1.1:	A parallel application represented as DAG	2
Figure 1.2:	A LAN can host a local Grid which can itself be a part of a global Grid	3
Figure 1.3:	A four layered architecture of Grid environment	5
Figure 1.4:	Classification of Grid scheduling algorithms	12
Figure 1.5:	Taxonomy of objective functions in Grid scheduling.....	13
Figure 1.6:	Classification of our research work on grid scheduling algorithms.....	18
Figure 2.1:	Gantt charts showing schedules of independent task scheduling algorithms.....	23
Figure 2.2:	A precedence constrained task graph.....	27
Figure 2.3:	(a) DAG and System; (b) HEFT schedule; (c) A better schedule.....	29
Figure 2.4:	(a) DAG; (b) and (d) Linear clustering; (c) and (e) Non-linear clustering	34
Figure 2.5:	(a) A clustered DAG and its CP shown in thick arrows; (b) The Gantt chart of a schedule; (c) The scheduled DAG and the DS shown in thick arrows	35
Figure 3.1:	SHCP scheduling algorithm	54
Figure 3.2:	Performance comparison of SHCP algorithm.....	56

Figure 3.3:	A simple DAG with precedence constraints	60
Figure 3.4:	Gantt charts showing the schedules of (a) HLD; (b) HED for DAG shown in Figure 3.3.....	63
Figure 3.5:	HED algorithm	64
Figure 3.6:	A simple DAG with precedence constraints	66
Figure 3.7:	Gantt charts showing the schedules of (a) SD; (b) RD for DAG shown in Figure 3.6.....	66
Figure 3.8:	Performance comparison of HED on random graph suite for heterogeneous systems.....	69
Figure 3.9:	Performance comparison of RD on random graph suite for homogeneous systems.....	71
Figure 4.1:	Effectiveness of duplication based scheduling over list scheduling	77
Figure 4.2:	A sample Grid consists of four resources.....	78
Figure 4.3:	A Grid resource management and scheduling model (GRMS).....	83
Figure 4.4:	The pseudo code for primary scheduling in SODA.....	84
Figure 4.5:	The pseudo code for secondary scheduling in SODA.....	85
Figure 4.6:	Effect of grid size on effective schedule cost.....	88
Figure 4.7:	Effect of workflow size on effective schedule cost	88
Figure 4.8:	Effect of workflow size on average NSL.....	89
Figure 4.9:	A bi-criteria optimization process	92

Figure 4.10: The pseudo code of secondary scheduling in DBSA.....	94
Figure 4.11: Gantt charts showing DBSA schedules (a) to (c) Primary scheduling; (d) Secondary scheduling with sliding constraints....	96
Figure 4.12: Performance comparison of DBSA algorithm	99
Figure 4.13: Real scenario of schedules of Figure 4.11 generated using Grid scheduling tool	101
Figure 5.1: Eigen-Trust algorithm	109
Figure 5.2: Relative schedules of DCA and Trust-MOS in primary scheduling.....	112
Figure 5.3: Primary scheduling algorithm of Trust-MOS	113
Figure 5.4: Secondary scheduling algorithm of Trust-MOS	115
Figure 5.5: Comparison of trustworthiness with respect to (a) number of tasks; (b) number of grid resources	117
Figure 5.6: (a) Comparison of trustworthiness for different sliding constants; (b) Comparison of trust cost overhead w.r.t. number of tasks; (c) Relative makespan w.r.t. number of resources.....	118
Figure 5.7: QoS based Grid scheduling system	127
Figure 5.8: Gantt chart showing schedule of AQuA	127
Figure 5.9: Gantt chart showing schedule of QGMM	129
Figure 5.10: Gantt chart showing schedule of QGMM–A	129

Figure 5.11: The pseudo code of AQuA algorithm	130
Figure 5.12: Performance in terms of availability surplus w.r.t. dedicated nodes	132
Figure 5.13: Performance in terms of makespan w.r.t. dedicated nodes	132
Figure 5.14: Performance in terms of availability surplus w.r.t. grid size	133
Figure 5.15: Performance in terms of makespan w.r.t. grid size	133
Figure 5.16: Performance in terms of QoS satisfaction w.r.t. grid size	134
Figure 5.17: Performance in terms of QoS satisfaction w.r.t. application size.....	135
Figure 5.18: Performance in terms of availability surplus w.r.t. application size.....	136
Figure 5.19: Performance in terms of makespan w.r.t. application size	136

LIST OF TABLES

Table 1.1:	World-wide utilization of IT resources.....	8
Table 2.1:	Evolution of scheduling algorithms with parallel and distributed system architecture.....	21
Table 3.1:	Computation Cost Matrix [ω_{ij}] for DAG in Fig. 3.3.....	61
Table 4.1:	Resource capacity.....	79
Table 4.2:	Machine price.....	79
Table 4.3:	Computation cost, b-level and task sequence for DAG in Figure 1.1.....	80
Table 4.4:	Grid simulation environment layouts.....	90
Table 5.1:	Grid environment layouts.....	116
Table 5.2:	Simulation parameters for schedules shown in Figures 5.8 to 5.10.....	128
Table 5.3:	Computation cost matrix for schedules shown in Fig. 5.8 to 5.10.....	128
Table 5.4:	Simulation setup parameters.....	131

1.1 Overview

It's hard to imagine modern day research without high performance computing systems. Scientists, economists and engineers who want to simulate complex processes need *High Performance Computing* (HPC). The emergence of high performance massively parallel computers has triggered the demand for development of new efficient scheduling algorithms which can take advantage of this technology [97]. A *parallel computer* is a multiple processor computer that enables the concurrent manipulation of data elements belonging to one or more processes for solving a single problem. A parallel computer can be categorized as (1) centralized multiprocessor, or (2) multicomputer [97]. A *centralized multiprocessor* (also known as symmetric multiprocessor or SMP) is a highly integrated system in which all processors communicate through a shared common global memory. On the other hand, a *multicomputer* system consists of several autonomous computers and an interconnection network, where the processors on different computers communicate by passing messages to each other. In current scenario, *Clusters* and *Grids* are the most common examples for multicomputer systems.

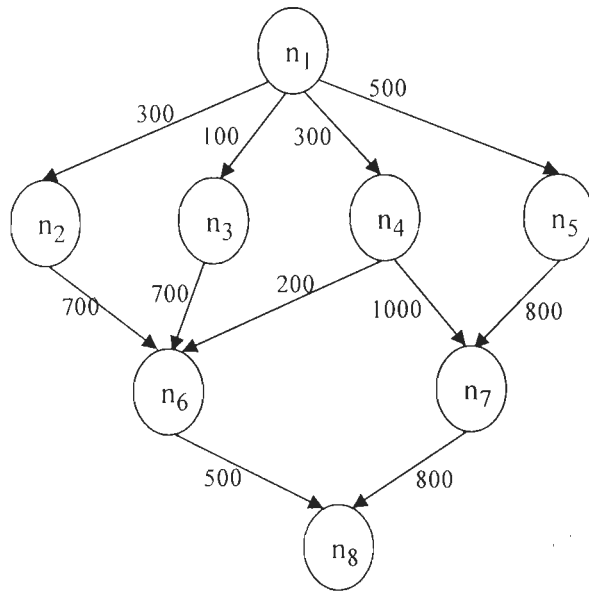


Figure 1.1: A parallel application represented as DAG (values are in Kbytes)

In general, a parallel application could be made up of multiple independent tasks or precedence-constrained tasks (also known as workflows). A *workflow* application is generally modeled as a directed acyclic graph (or DAG) as shown in Figure 1.1, where nodes represent the application tasks and edges represent inter task data dependencies [144]. The task scheduling in multiprocessor computing environment is NP-hard problem [65, 72]. NP may be equivalently defined as the set of decision problems that can be solved in polynomial time on a nondeterministic Turing Machine.

1.2 Grid Computing

The purpose of a computation Grid (the term derives from electrical “power grid”) is to provide transparent access to resources such as systems, data, applications and services via Internet. The idea was that accessing compute power from a Grid would be as simple as accessing electrical power from an electrical grid. This kind of Grid delivers an effective and efficient way to

perform complex tasks over the resources present anywhere regardless of the organizational boundaries. The most obvious resource included in the Grid is a processor, but Grids also encompass sensors, data-storage systems, catalogs, applications and network resources, etc. A computing Grid spans domains of different dimensions, starting from local Grids made up of resources linked through LANs, up to global Grids made up of heterogeneous resources owned by different organizations connected over Internet (Figure 1.2).

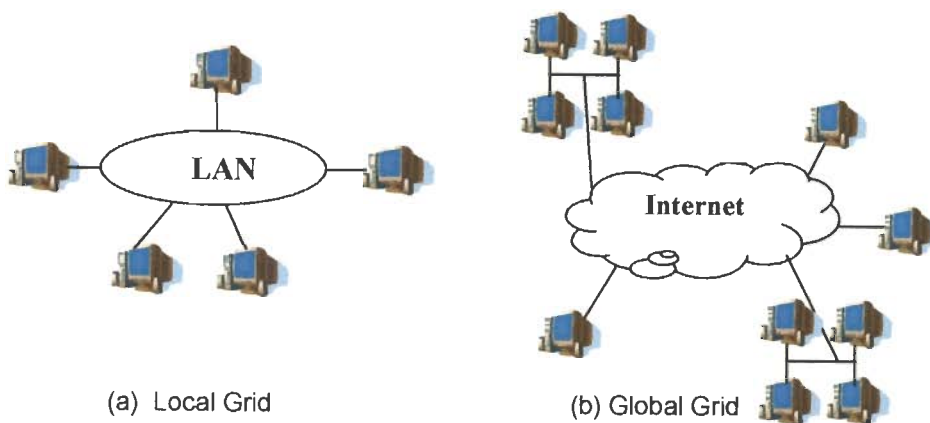


Figure 1.2: A LAN can host a local Grid which can itself be a part of a global Grid

One of the first commonly known GRID initiatives was the SETI@Home [112], which solicited several million volunteers to download a screensaver that used idle processor capacity to analyze data in the search for extraterrestrial life. In SETI (Search for Extraterrestrial Intelligence) @Home project, thousands of people are sharing the unused processor cycles of their PCs in the vast search for signs of "rational" signals from outer space. In a more recent example, the Tele Science Project [123] provides remote access to an extremely powerful electron microscope at the *National Center for Microscopy and Imaging Research* in San Diego. The authorized users of the Grid can remotely operate the microscope, allowing new levels of access to the instruments and their capabilities. Many scientific Grids are appearing to investigate problems such as galactic evolution, climate modeling, aircraft design, urban planning, molecular dynamics and animation [26, 83].

By definition [44], “A *computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high end computational capabilities*”. According to Foster’s check list, the minimum expected properties from a Grid system are:

- It coordinates resources that are not subject to centralized control (i.e., resources owned by different organizations or under the control of different administrative units) and at the same time addresses the issues of security, policy, payment, membership and so forth that arise in these settings.
- It uses standard, open, general purpose protocols and interfaces. These protocols address fundamental issues such as authentication, authorization, resource discovery and resource access.
- It delivers nontrivial quality of service i.e., it is able to meet complex user demands, e.g., response time, throughput, availability, security, etc.

1.2.1 Grid Architecture

The Grid computing infrastructure may be visualized as a four layered architecture as shown in Figure 1.3. Some companies may have products that span all tiers of this architecture, while others may be specifically focused on a particular niche. It is important to realize that grouping products and services into categories in this manner is far from exact.

Fabric Ware: The fabric layer in this architecture refers to hardware products that are used to build the Grid infrastructure. These include personal computers, clusters, high performance systems, storages and networking devices such as routers, switches, load balancers, caching systems, etc. The fabric layer also consists of various types of sensors and other scientific instruments. Companies such as IBM, Hitachi, Fujitsu, Sun Microsystems, HP,

Cisco Systems and many others can be categorized as fabric providers in this taxonomy.

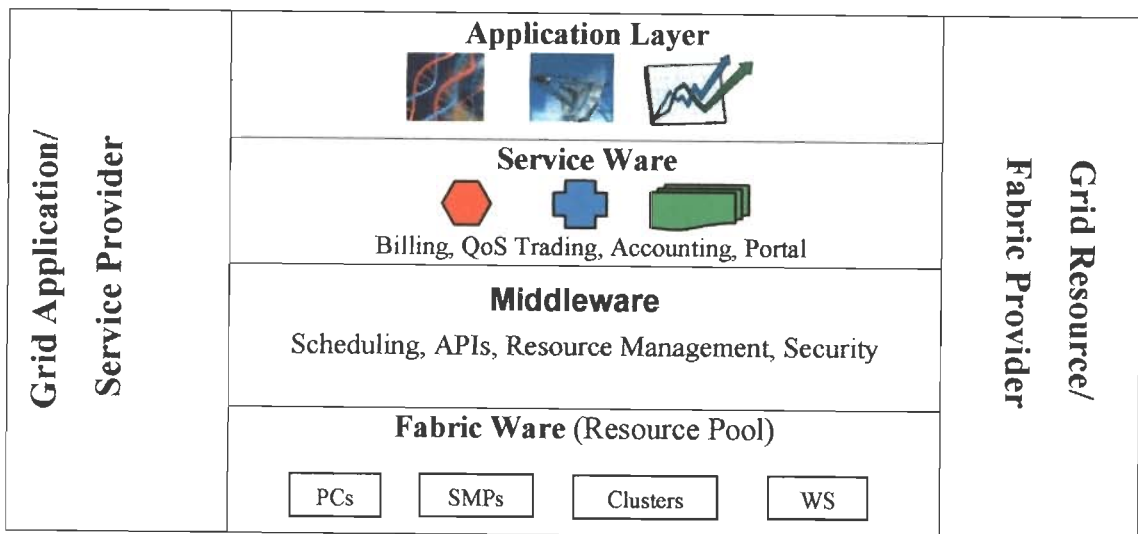


Figure 1.3: A four layered architecture of Grid environment

Middleware: The middleware layer consists of software and products that manage and facilitate access to the resources available at the fabric layer. These products perform the function of resource management, security, scheduling, and execution of tasks. The middleware layer also includes the tools developed to facilitate use of resources by various applications. A parallelization tool or compiler would reside in this layer of the matrix. Middleware companies include Avaki, United Devices, Entropia, Grid Systems, Tsunami Research, and Platform Computing, etc. Additionally, software companies that help in the development of applications for the Grid, such as GridIron Software and Engineered Intelligence, also fall into this category.

Service Ware: The service layer consists of services that provide the operational support systems for grid computing. These include billing, QoS trading, accounting and management software. Companies such as

Gridfrastructure, GridXpert and Montague River provide products and services that fall within this layer.

Application Layer: The application layer consists of all the software applications and services that utilize the Grid infrastructure. There are thousands of companies that will eventually adapt their applications for the Grid. Some early adopters include Wolfram Research (makers of Mathematica), Lion Biosciences, Fluent Inc, Accerlys, Cadence, Mentor Graphics, and others.

Grid Resource Provider: The Grid Resource Provider (GReP) is an entity that provides the various resources listed in the fabric layer, middleware and service ware layers as services that can be leased on variable terms. For example, a customer may rent CPU cycles from a grid resource provider while running an especially compute intensive application, while at other times he/ she may want to rent additional storage capacity. The GReP is not aware of the applications being executed by the customer on its platforms. Grid Service Providers today include companies such as Gateway Corporation, ComputingX, IBM, and NTT Data, etc.

Application Service Provider: The Grid Application Service Provider (GASP) provides end-to-end Grid computing services to the user of a particular application. The customer in this case will purchase “application time” from the provider and will provide the data or parameters to the GASP through an application portal. In future, it may also be done through published web service specifications. The GASP may choose to purchase services from the GReP or may choose to build the infrastructure organically. Grid computing is powering science across the globe, providing the technology to explore new ways of doing science. Through Grid, scientists can now share data, data storage space, computing power and results. Collectively researchers can tackle bigger questions than ever before: from disease cures and disaster management to global warming and mysteries of the universe.

1.2.2 Grid Computing in Research

Computational Grids in research projects around the world are aiming at higher and higher throughputs e.g., SETI@Home computes at over 730 TFLOPS as of April 2010 [112], MilkyWay@Home computes at over 1.6 PFLOPS as of April 2010 [86], Einstein@Home is crunching more than 210 TFLOPS as of April 2010 [40] and Folding@Home computes at over 5 PFLOPS, as of March 17, 2009 [43]. Many of these e-Science enabling projects would be impossible without massive computing power of such Grids. Scientists are using Grids in various fields such as:

- **Biologists** are using Grids to simulate thousands of molecular drug candidates on their computers, aiming to find a molecule able to block specific disease proteins [39, 43, 120].
- **Earth scientists** are using Grids to track ozone levels using satellites, processing hundreds of Gigabytes of data every day (the equivalent of about 150 CDs a day!) [28].
- **High energy physicists** are using Grids in search for a better understanding of the universe, relying on tens of thousands of desktops to store and analyze the 10 Petabytes of data (equivalent to the data on about 20 million CDs!) produced by the Large Hadron Collider (LHC) each year. Thousands of physicists in dozens of universities around the world want to analyze this data [40, 86, 112].
- **Engineers** are using Grids to study alternative fuels, such as fusion energy [48].
- **Artists** are using Grids to create complex animations for feature films [21, 70].

1.2.3 Grid Computing in Business

Today, business organizations depend on information technology (IT) more than ever before in the past. Trillions of dollars have been spent in the last decade by them to improve efficiency and optimize all aspects of their operations, such as financial, audit, supply chain, back office, sales and marketing, engineering, manufacturing and product management. Most of the investments have been made in technology to achieve these objectives. Despite this, much of the IT infrastructure remains underutilized (Table 1.1) [61].

Table 1.1: World-wide utilization of IT resources

IT Resource	Average Utilization / Day
Windows Servers	< 5 %
Unix Servers	15 – 20 %
Desktops and Workstations	< 5 %

To solve new complex and challenging problems being faced/ discovered, massive computational power is required. The underutilized computing power available in the existing IT infrastructure may be harnessed using Grid technologies for solving such problems. Grid computing is harmonious amalgamation of numerous technologies to provide a valuable and meaningful service to users. Now, Grid computing has entered into the commercial market after successfully developing its roots in the global academic and research communities. Grid computing is offering low cost and high throughput computing solutions to companies through optimizing and leveraging existing underutilized IT infrastructure and investments as shown in the Table 1.1. Grids deployed on exiting infrastructure provide 93% savings in up-front hardware cost and mitigate the need for additional investments in hardware, software and maintenance costs, etc. The Grid can be deployed as quickly as in two days, with little or no disruption to operations while cluster takes 60–90 days to deploy in an enterprise. It is analyzed that the operational

expenses in Grid deployment are 73% less than similar HPC based solution. Grid exhibits scalability in adding new resources without increasing functional complexity.

With the growing complexities of the applications, many *Independent Software Vendors* (ISV) are realizing that the cost of maintaining and managing such applications is deterring clients from purchasing them. For example, Fluent, the world's largest computational fluid dynamic software company has started to outsource its products via *Application Service Provider* (ASP) model. Their software had an initial cost of US \$150,000 and US \$28,000 per user licensing cost, but they are now offering their product for US \$15.00 per CPU/Hour. Now, there is no hardware, software and maintenance costs to the customers. Similar to ASP model, *Grid utility* model allows customers to pay-per-use basis for the amount of computational, storage or network capacity they use. It helps in reducing the IT costs up to 40–50% in many cases [15].

In drug research, Grid computing is being used in the drug discovery phase to screen suitable drug like molecules against diseases and also for clinical simulation, healthcare ecosystem modeling and pharmacokinetic simulations. In short, grid computing is allowing drug companies to get the most out of their R&D expenditure by developing the right product and getting it to market in the shortest possible time. Companies can save almost US \$5M per month in R&D expenses for each month shaved off the drug development process [34].

Today, large corporations such as IBM, Sun Microsystems, Intel, Hewlett Packard and some smaller companies such as Platform Computing, Avaki, Entropia, DataSynapse and United Devices are putting their marketing dollars to good use and creating the next generation of thought-leadership around grid computing that is focused on business applications rather than academic and basic research applications. High performance computing statistics are

extremely important because they tell us which industries already have demand for tremendous computing power.

The vision of Grid computing can be implemented by the use of open standards with common interfaces and protocols for defining, discovering and using the heterogeneous set of resources. The Global Grid Forum (GGF) [122] is an organization that has undertaken the mission of creating and documenting the technical specifications and implementation guidelines that promote and support the development, deployment and implementation of Grid technologies and applications. Specifically, GGF has published a standard Open Grid Services Architecture (OGSA) and an Open Grid Services Infrastructure (OGSI 1.0) based on OGSA. The major goal of grid technology is to promote a virtual computing environment, which allows access to a set of services through transparent and coordinated use of distributed and heterogeneous resources. Services are the abstraction of resources and they can be computing cycles, software, documents, data, storages, and so on.

The time and accuracy of various compute-intensive and data-intensive tasks are limited by the availability of the resources within the silos. Grid computing allows a pool of heterogeneous resources both within and outside of an organization to be virtualized and form a large virtual parallel computer. This virtual parallel computer can be used by a collection of users and/or organizations in collaboration to solve their problems. This use of a virtual computer formed from a Grid of shared resources allows the users of the virtual organizations to solve complex collaborative problems. This enhanced sharing allows users and organizations to improve performance, reduce cycle time, increase availability and improve fault tolerance by distributing both compute-intensive and data-intensive workloads across several resources forming the Grid. Further, the sharing of the resources from the individual silos forming the bigger pool allows the underutilized and available resources to be discovered and should be used by applications and jobs that have a greater need. This not

only improves the utilization of underused resources, but also improves the performance and availability of resource-intensive jobs and applications.

1.3 Grid Scheduling and Research Issues

The success of various applications in making effective use of underutilized resources is greatly dependant on proper scheduling of application tasks on grid resources. As a result, scheduling has become one of the most active areas of research in the Grid community. The problem of scheduling in the Grid environment is NP-hard, and there is always a possibility of improvement and optimization. *Scheduling* is a decision making process by which application components are matched and assigned to the available resources to optimize the various performance metrics while respecting the service level agreements (SLAs).

By definition [95], "*Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has a goal of optimization of one or more objectives.*" The scheduling process may be seen as consisting of three stages: (1) resource discovering and filtering, (2) resource selecting and scheduling according to certain objectives, and (3) job submission [111]. In general, scheduling algorithms concentrate on the second stage i.e., resource selection and scheduling. The resources may be robots in manufacturing process, machines in a workshop, runways at an airport, programmers in a software development project or processing nodes in a computing environment and so on [55]. To convey this generalization, the term "*resource*" is usually used in place of processor. Similarly, the tasks may be operations in a manufacturing process, transportations of items from town A to town B, take-offs and landings at an airport, execution of computer programs and so on. A metatask (or a set of tasks) must be divided into subtasks which then have to be scheduled onto the resources while satisfying an objective or a

set of objectives as defined by the given performance measure. The objectives may be the minimization of the completion time of the last task in a set of (dependent) tasks, minimizing the average completion time in a set of (independent) tasks, minimization of the economic cost of executing tasks on resources, maximizing the resource utilization, etc. There exist several scheduling algorithms that fall in different categories of the Grid scheduling taxonomy as illustrated in Figure 1.4.

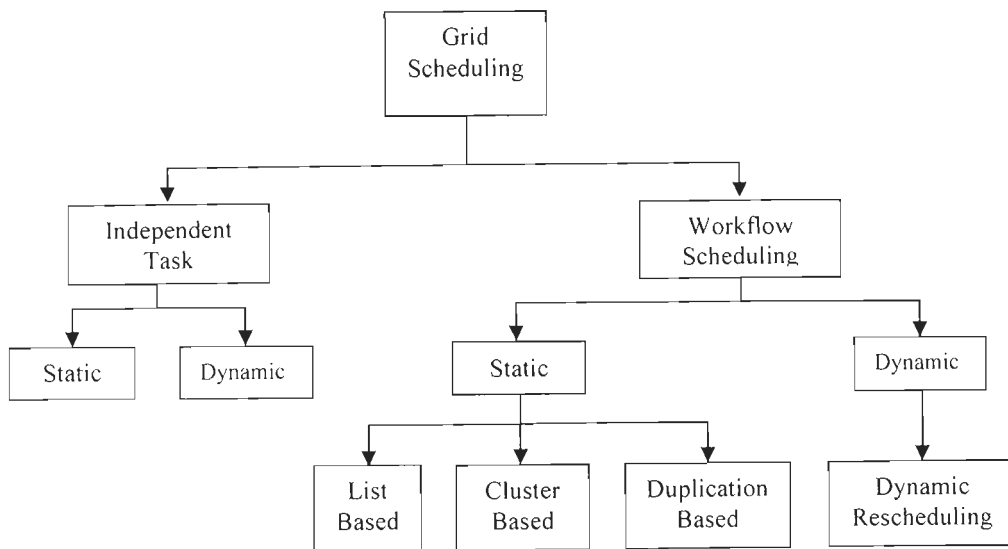


Figure 1.4: Classification of Grid scheduling algorithms

In *independent task scheduling (or job scheduling)*, a grid application is composed of a set of independent tasks (referred as jobs), whereas in *workflow scheduling*, the same is modeled as workflow (Figure 1.1). A *workflow* is a set of dependent tasks with precedence constraints that have to be executed in a well defined order to achieve a specific goal [56, 113]. Scheduling algorithms for workflow applications in grid systems can be classified into three categories, namely, list based, clustering based, and duplication based algorithms as depicted in Figure 1.4. Depending on the time at which the scheduling decisions are made; the algorithms can be static or dynamic. In static scheduling [136], information regarding the resources available in the Grid and the tasks in an

application is assumed to be known a priori before scheduling, whereas, in dynamic scheduling [11, 47, 71, 88], task allocation is done on the fly as the application executes. Both static and dynamic scheduling algorithms are widely adopted in Grid computing.

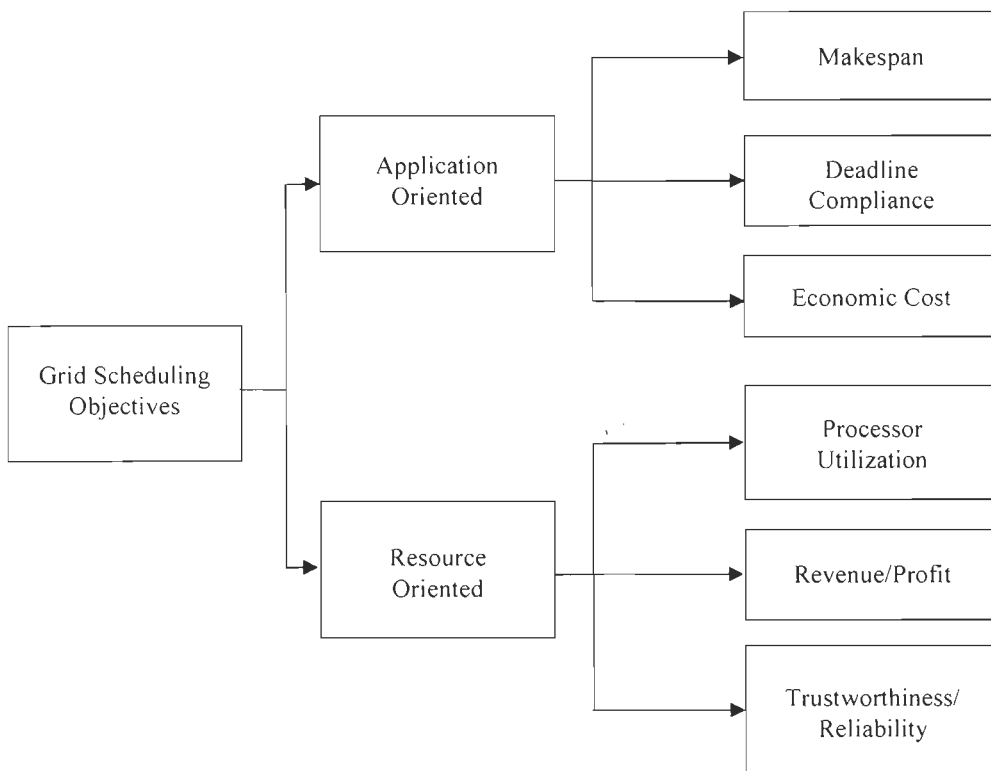


Figure 1.5: Taxonomy of objective functions in Grid scheduling

The current research challenges in grid scheduling are to optimize one or more objectives while respecting constraints to the level necessary. There exist many different performance objectives to be optimized in grid scenario (Figure 1.5). As said earlier also, the objectives may be the minimization of makespan or economic cost of schedule of an application, or maximization of processor utilization, trustworthiness and reliability of resources, etc. There is a shift in research towards developing scheduling algorithms that meet two or more objectives at the same time during optimization phase. For example, scheduling may attempt to map an application on a set of resources in order to achieve

good throughput and high reliability. However, the optimization of one may be at a lower cost to others. Most part of the current research in literature on scheduling problems is dedicated to single criterion approach whereas, the use of multiple criteria is expected to provide a more realistic solution for the decision maker.

The Foster's vision, discussed in Section 1.2, is now becoming a reality as follows:

- The user submits his/her request through a *Graphical User Interface* (GUI) just specifying the high level requirements (the kind of application he wants to use, the operating system, etc.) and possibly providing input data.
- The Grid finds and allocates suitable resources (computing systems, storage facilities, etc.) to satisfy the user's request.
- The Grid monitors request processing.
- The Grid notifies the user when the results are available allowing their retrieval.

However, due to the dynamic changes in grid resource properties and user requirements, the Grid will have to face the following situations:

- Task failure, random arrival of high priority tasks.
- Resource failure, joining of new resources, imbalance in workload on resources, changing resource properties (number of users, cost, etc.).
- Changing task properties (priority, deadline, dependencies, etc).

Further, many characteristics unique to grid situation make the design of scheduling algorithms more challenging [90], as seen below:

- **Autonomy** – Resources in grids are autonomous and the grid scheduler does not have full control over the resources. The owners of these resources are free to establish and implement their own policy regarding scheduling, security, etc. Thus, a *Grid Scheduler* is required to be adaptive to different local policies.
- **Heterogeneity** – Grid resources like processor architectures, operating systems, networks, applications tools are heterogeneous and distributed in multiple domains over the Internet. The heterogeneity results in different capabilities for job processing and data access. The Grid must define standards, open protocols and interfaces to interact with these resources so that heterogeneity gets hidden.
- **Dynamism** – The Internet environment is continuously changing and the status of grid resources may vary with time. The scheduling algorithm should take such dynamic behaviors into account.
- **Fault Tolerance** – Resources may fail at any moment. Robustness with respect to failure of network, machines, software components, and so on is a critical issue. The scheduling algorithm should be fault tolerant to neutralize the effect of resource failure.
- **Trustworthiness** – Grid is an open distributed computing infrastructure. When defining a grid, users must be recognizable and the access to resources must be controlled. The resources forming the Grid must be trustworthy.
- **Scalability** – Grid performance must not be affected with the increase in the number of resources as well as users when it is operative.

During the past decade, the trends indicated by ever faster networks, distributed systems and multiprocessor computer architectures (even at the desktop level) clearly show that *parallelism is the future of computing*. The MPI

(Message Passing Interface) [87] and PVM (Parallel Virtual Machine) [50] environments make it possible to write one distributed task (or application) which runs on a distributed system like Cluster or Grid. Scheduling is a decision making process concerned with the allocation of resources to the application tasks over specified time durations in order to optimize one or more performance metrics [95].

In the late 1960s, computational resources (such as CPU, memory and I/O devices) were scarce. Efficient utilization of these scarce resources could lower the cost of executing computer programs. This provided the economic reason for the study of scheduling. It has been observed by several scientists that a large number of resources (quantity) contributing to a Cluster or Grid cannot be coordinated to execute an application in an optimal or sub-optimal way (quality) without using a well developed scheduling strategy.

1.4 Motivations and Objectives

In 1960s, *scheduling* became two separate research areas, with people looking at the scheduling of processes, jobs, etc onto CPUs and computers separating off from people concerned with optimizing timetables, organizing factory workflows, etc. Scheduling on the Grid presents an excellent opportunity to reunite these long separated siblings.

Today, different communities with different requirements and objectives have arrived in the Grid community. People need to be able to dispatch work, but with an assurance about when the work will be completed, performance and cost, etc., all negotiated as a part of some service level agreement (SLA). An SLA is a contract between a resource provider and a client specifying the quality of service (QoS) that can be expected (i.e., the time by which a task will be dispatched). These contracts enable both the parties to reach a mutually satisfactory agreement. So the designer of a Grid based scheduler need to

analyze “How can we best schedule this work onto grid resources, given that each job has an agreed set of constraints, so that we meet as many constraints as possible while still trying to maximize income?”. Grid scheduling is the mapping of individual tasks to compute resources, while respecting SLAs, etc. A variety of optimization criteria are of interest in grid scheduling: minimization of maximum lateness, minimization of economic cost to the user, maximization of resource utilization, fairness, minimization of variance, maximization of resource availability, and trustworthiness, etc.

As scheduling in Grid is NP-hard problem, there is no clear choice of method which will be the best for grid scheduling. This motivates us to design new scheduling heuristics for Grid computing systems. A heuristic is a technique that seeks good solution at a reasonable computation cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is [103].

Real world scheduling problems are heterogeneous. Heterogeneous environment may use different data and models, and operate in different modes. Most recently, the economic approaches have gained popularity in Grids to satisfy certain criteria like minimization of economic cost to the user, rather than seeking a solution that offers the best performance with minimization of execution time. In view of this, we intend to propose new scheduling techniques for optimizing multiple objectives in grid scheduling.

Although task scheduling in parallel and distributed systems has been extensively studied, new challenges in grid environments still make it an interesting topic, and many research projects are underway. Through our study on current scheduling algorithms working in the grid computing, we can find heterogeneity, dynamism, computation and data separation, trustworthiness, and fault tolerance as primary challenges for current research in Grids. QoS is the concern of many grid applications. Most current research concentrates on how to guarantee the QoS requirements of the applications, but few of them

study how the QoS requirements affect the resource assignment and subsequently the performance of the other parts of the applications. Situation becomes more complex when there are more tasks having QoS requirements.

In grid computing, two major security issues are often encountered. First, user programs may contain malicious codes that may endanger or weaken the grid resources assigned. Second, shared grid resources, once infected by network attacks or malicious resources, joining the grid may damage the applications. The research in grid security primarily focuses on existing cryptographic based mechanisms for the protection of grid resources only. However, the protection of user applications remains a challenging issue as the grid resources have the ultimate power of controlling the execution of user programs or task requests.

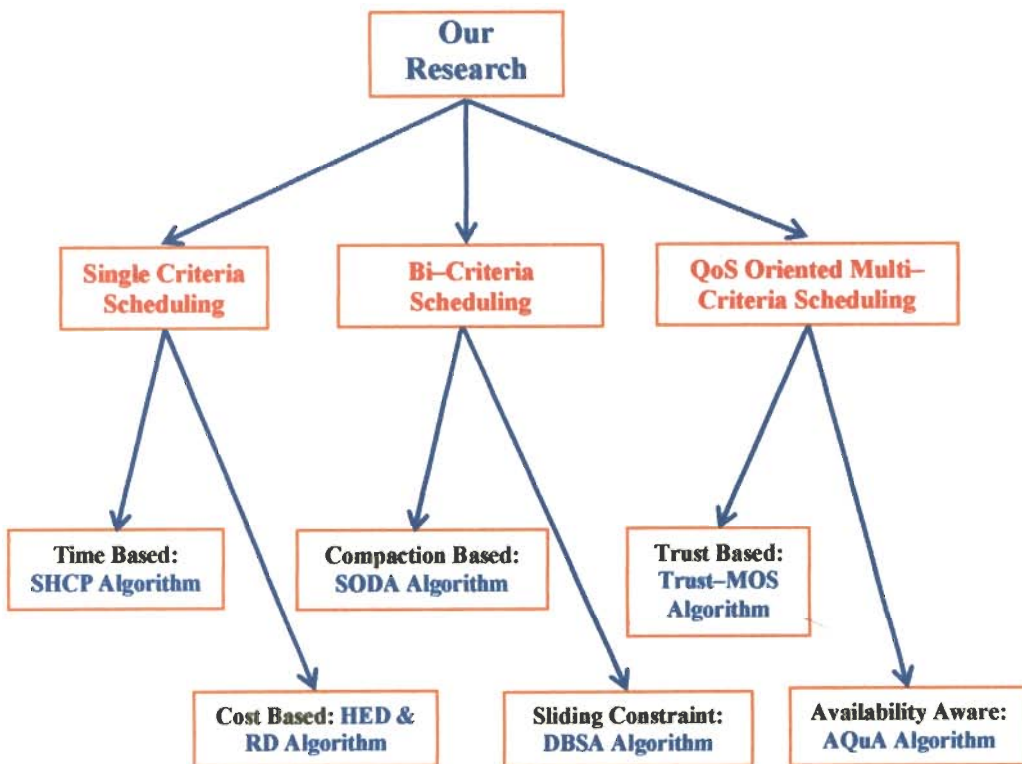


Figure 1.6: Classification of our research work on grid scheduling algorithms

The computational Grid usually has abundant computational resources (number of resources is unbounded in some duplication algorithms), but high communication cost. This can make task duplication very graceful and cost effective in dynamic and heterogeneous grid environment. We plan to exploit duplication in selective manner so that schedule length/cost is minimized without incurring unreasonable overhead costs.

In summary, the objective of this research work is to develop new scheduling strategies and validate them through simulation such that they meet multiple criteria, ensure QoS requirements and secure the applications through selecting trustworthy resources.

1.5 Organization of Thesis

The next chapter discusses several scheduling heuristics proposed by different researchers in the past. Figure 1.6 shows our research work which can be classified into three categories namely, Single criteria scheduling, Bi-criteria scheduling, and QoS Oriented Multi-criteria scheduling. In single criteria scheduling, time-based and cost-based heuristics are proposed while in multi-criteria scheduling optimization of both type of objectives has been attempted. In Chapter 3, the matter of heterogeneity and effectiveness of duplications for single criterion scheduling in heterogeneous grid environments are discussed. Chapter 4 deals with scheduling heuristics for multiple criteria scheduling problems. This research work is extended considering different QoS constraints like availability and trustworthiness of grid resources in Chapter 5. This chapter presents QoS based scheduling heuristics for grid computing systems. The concluding part of this research work is summarized in Chapter 6 indicating our contributions and proposes future research directions in multi criteria scheduling for multiple workflows that needs more attention by researchers in future.

There are three main players in the scheduling problem: computational resources/communication network, scheduler, and application tasks. In this chapter, we have discussed some earlier research work that has been done towards developing scheduling algorithms for optimized performance.

2.1 Review of Scheduling Strategies

As already discussed, scheduling is crucial for an effective utilization of Grid computing system. In general, scheduling methodology will differ depending on whether the application to be scheduled is made up of independent tasks or multiple interactive tasks (dependent tasks). Dependent tasks are characterized by precedence constrained relationship among tasks i.e., a task can not start until all of its predecessors are completed. Task dependency has crucial impact on the design of scheduling algorithms. Further, scheduling can be static or dynamic (Figure 1.4) [146]. Static scheduling requires a priori knowledge about tasks and their mapping on suitable

resources is done before execution begins which remains unchanged till the end. In dynamic scheduling, the mapping (or re-assignment) of tasks is possible during runtime. In addition, some hybrid techniques (static-dynamic hybrid scheduling) may be designed to take advantage of static schedule, and at the same time capture uncertain behavior of applications and resources. For example, some tasks in an application may have QoS requirements. The static approach is used to schedule these QoS tasks and dynamic scheduling is applied to the rest of tasks.

In literature, scheduling algorithms have been intensively studied as basic problem in traditional parallel and distributed computing systems, such as symmetric multiprocessor machines (SMPs), massively parallel processors systems (MPPs), and cluster of workstations (COW) [47]. It can be concluded that scheduling algorithms are evolving with the architecture of parallel and distributed computing systems. In Table 2.1, we are exploring some important features of parallel and distributed systems and typical scheduling algorithms adopted.

Table 2.1: Evolution of scheduling algorithms with parallel and distributed system architecture

<i>Architecture</i>	SMP, MPP	COW	Grid
<i>Chronology</i>	Late 1970s	Late 1980s	Mid 1990s
<i>Interconnection Network</i>	Bus, Switch	LAN, ATM	WAN/ Internet
<i>Cost of Interconnection</i>	Very Low/Negligible	Low but not negligible	High
<i>Network Heterogeneity</i>	None	Low	High
<i>Compute node heterogeneity</i>	None	Low	High
<i>Single System Image</i>	Yes	Yes	No
<i>Resource Pool</i>	Predetermined	Predetermined	Not Predetermined
<i>Nature of Resource Pool</i>	Static	Static	Dynamic
<i>Resource Management Policy</i>	Monotone	Monotone	Diverse
<i>Scheduling Algorithms</i>	Homogeneous Scheduling	Heterogeneous scheduling	Grid Scheduling

The traditional scheduling algorithms cannot be directly fitted in the Grid computing environment because they produce poor schedules. The assumptions made by traditional algorithms do not hold in grid circumstances.

In [25], Casavant and Kuhl proposed a scheduling taxonomy for general purpose parallel and distributed systems. Since Grid is a special kind of such systems, the scheduling algorithms for Grid form a subset of general scheduling taxonomy as shown in Figure 1.4 [129].

2.1.1 Independent Task Scheduling

A set of independent tasks is assigned according to the capacity of resources in order to achieve high system throughput. In [19], Braun et al. provided a comparison of 11 static heuristics for scheduling in heterogenous computing environment in which load balancing was main objective. Some of these are briefly discussed below:

OLB (Opportunistic Load Balancing): It assigns each task in an arbitrary order to the processor that is having the shortest schedule, irrespective of the *ETC* (expected time to compute) on that processor. Thus, it tries to balance load on processors, but generates a poor schedule. In Figure 2.1 (b), OLB schedule is illustrated using Gantt chart. An expected time to compute matrix is represented in Figure 2.1 (a) which shows the different capabilities of tasks on different processors. The makespan of a schedule obtained using OLB is 19.

MET (Minimum Execution Time): It assigns each task in an arbitrary order to the processor on which it is expected to be executed fastest regardless of the current load on that processor. In other words, it chooses the fastest processor for the task under consideration. It may generate a better schedule than OLB, but certainly leads to poor load distribution among processors. Figure 2.1 (c) shows a schedule of makespan 17 generated with MET approach. The MET schedule is better than OLB as it tries to schedule tasks over fastest resources to minimize the execution time.

(a) ETC (Expected Time to Compute) Matrix

	Processor 1	Processor 2
Task 1	3	5
Task 2	8	11
Task 3	11	8
Task 4	6	8

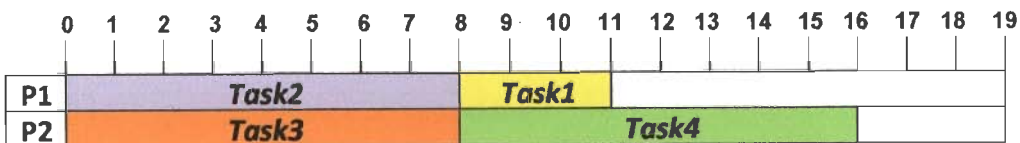
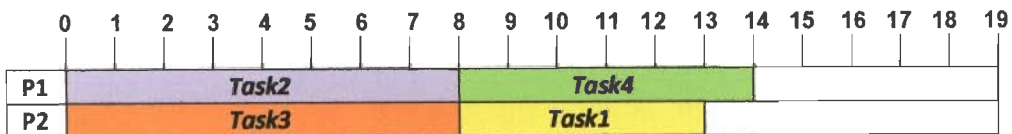
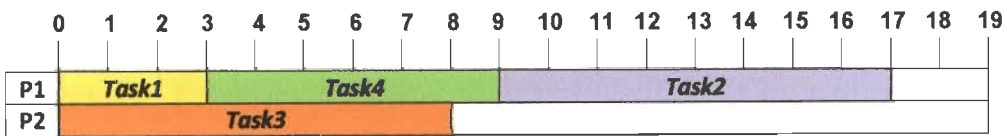
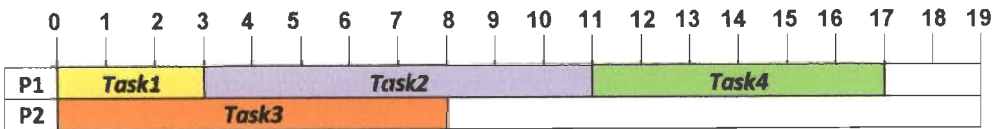
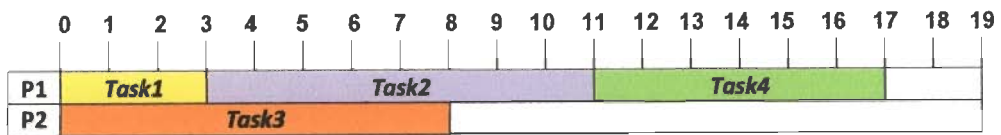
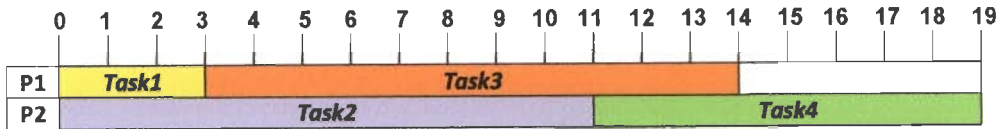


Figure 2.1: Gantt charts showing schedules of independent task scheduling algorithms

MCT (Minimum Completion Time): It assigns each task in an arbitrary order to the processor with expected minimum completion time. This causes some tasks to get assigned to slower machines. The intuition behind MCT is to combine the benefits of OLB and MET. A schedule is obtained using MCT approach (Figure 2.1 (d)). The schedule length (makespan) is 17, which is same as MET but it tries to assign tasks in order to minimize overall completion time to shorten the schedule length.

Min–Min: It does not consider tasks in an arbitrary order. It first determines the MCT for each unscheduled task and then assigns the task with minimum MCT (Min–Min) to the processor which offers it this Min–Min time. It can produce shorter schedule with better load balancing than MCT. In [60], a QoS Guided Min–Min heuristic has been presented which can guarantee the QoS requirements of particular tasks and minimize the makespan at the same time. Figure 2.1 (e) shows a schedule obtained using Min–Min approach.

Wu et al. [134] gave a Segmented Min–Min algorithm, in which tasks are first ordered by their expected completion time (it could be the maximum *ECT*, minimum *ECT* or average *ECT* on resources), then the ordered sequence is segmented, and finally Min–Min is applied to all these segments. The segmentation improves the performance of typical Min–Min when the lengths of the tasks are dramatically different by giving a chance to longer tasks to be executed earlier than in the case where the typical Min–Min is adopted.

Gao et al. [49] presented an *Estimation Based Grid Scheduling Mechanism* (EBGSM) which allows the simultaneous running of local and grid tasks. They apply this method to MCT and Min–Min algorithms and show that EMCT and EMin–Min outperform MCT and Min–Min in respect of shorter makespan. But they applied the linear estimation mode which is not suitable to dynamic grid environment. Therefore, other estimation mode should be identified which are adaptable to the Grid to produce better schedules.

Max–Min: It is similar to Min–Min, but the task with maximum MCT (Max–Min) is assigned to the processor in order to avoid load imbalance. It allows larger jobs to schedule earlier than the smaller ones. Max–Min heuristic may give a mapping with a more balanced load across machines and a better makespan. Figure 2.1 (f) shows a Max–Min schedule of makespan 14 which outperforms the other heuristics.

XSuffrage: One more popular static heuristic for independent task scheduling is called Suffrage [85]. For a task, its suffrage value is defined as the difference between its best MCT and its second best MCT. Tasks with high suffrage value take precedence in scheduling. In this way, suffrage policy tries to minimize task suffrage. But when task has I/O data requirements, tasks should be assigned to the resources as near as possible to the data source to reduce the overall makespan.

If the resources are clustered and nodes in the same cluster have near identical performance, then the best and second best MCTs are also nearly identical which makes the suffrage close to zero and gives the tasks low priority. Other tasks might be assigned on these nodes so that the task might be pushed away from its data source. To fix this problem, Casanova et al. [23] gave an improvement called XSuffrage, which gives a cluster level suffrage value to each task. Figure 2.1 (g) shows a schedule obtained using XSuffrage.

The main issue in independent task scheduling is load balancing across the available processors. Any algorithm that focuses on this matter is likely to produce a better schedule. This is clearly illustrated in the example above (Figure 2.1) where Max–Min because of its focus on load balancing gives the better schedule.

Task Replication: In [114, 117], two algorithms have been proposed that do not use performance estimate but adopt the idea of duplication, which is also feasible in the grid environment where computational resources are usually abundant but mutable. Subramani et al. [117] introduced a simple distributed

duplication scheme for independent job scheduling in the Grid. Each job is distributed to the K least loaded sites where each of these K sites schedules the job locally. When a job is able to start at any of the sites, the site informs the scheduler at the job–originating site, which in turn contacts the other $K - 1$ sites to cancel the jobs from their respective queues. This improves system utilization and reduces average job makespan. The parameter K can be varied depending upon the number of resources in the grid.

Silva et al. [114] proposed a resource information free algorithm called *Work Queue with Replication* (WQR) for independent job scheduling in the Grid. The WQR algorithm uses task replication to cope with the heterogeneity of hosts and tasks, and also the dynamic variation of resource availability due to load generated by other users in the Grid. In WQR, an idle resource will replicate tasks that are still running on other resources. Tasks are replicated until a predefined maximum number of replicas are reached. When a task replica finishes, other replicas are cancelled. In this approach, performance is increased in situations when tasks are assigned to slow/busy hosts because when a task is replicated, there is a greater chance that a replica is assigned to a fast/idle host. Another advantage of this scheme is that it increases the immunity in performance variation, since the probability that all sites are changing is much smaller than one site [57].

Ritchie and Levine [104] proposed a static scheduling algorithm for independent jobs in heterogeneous computing environments using *Ant Colony Optimization* (ACO) techniques. This algorithm can be applied with some modification on workflows in dynamic grid environment which may give more realistic results.

Zhang and Luo [141] proposed a QoS group guided grid scheduling algorithm with task replicas which makes scheduling decision on the basis of recent QoS status feedback of resources for independent tasks. This algorithm divides the tasks into groups, and then schedules them in groups to different

resources. Later on, first running task that has not finished execution is replicated to the available resource. After finishing this task, its replicas on other resources are cancelled. This algorithm can be applied in DAG scheduling of tasks in grid computing environments. He et al. [60] proposed QoS Guided Min-Min (QGMM) scheduling algorithm to achieve high system throughput while matching applications with the available computing resources. This heuristic concerns over QoS requirement of tasks, i.e., network bandwidth. Tasks are classified as tasks with QoS requirements and tasks without QoS requirements. Tasks with QoS requirements get priority in scheduling.

2.1.2 Workflow Scheduling

Workflows are constituted by multiple tasks having precedence constraints, thus forming a directed acyclic graph (DAG) with node and edge weights (the nodes represent the tasks and the directed edges represent the execution dependencies as well as the amount of communication required) [65]. Figure 2.2 shows an example of precedence constrained task graph where task n_4 cannot start before tasks n_1 , n_2 and n_3 have finished. The scheduling objective is to minimize the program completion time under these constraints.

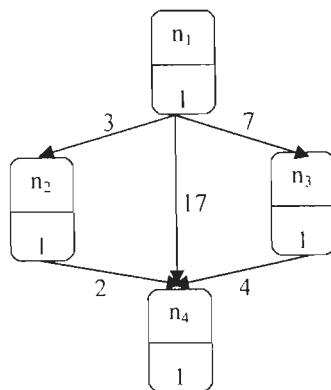


Figure 2.2: A precedence constrained task graph

There is a trade-off between taking maximum advantage of parallelism and minimizing communication delay. This problem is also called the Max-Min problem [41]. High parallelism means dispatching more tasks simultaneously to different resources, thus, increasing the communication cost, especially when the communication delay is very high. However, clustering tasks only on a few resources means low exploitation of parallelism. To deal with this problem in heterogeneous computing systems, three kinds of heuristics for static scheduling have been proposed: (a) List based heuristics [10, 29, 58, 84, 90, 98, 106, 124, 136], (b) Duplication based heuristics [5, 7, 9, 27, 37, 69, 73, 80, 91, 92, 93, 94, 102, 114, 117], and (c) Clustering based heuristics [51, 82, 135].

List based scheduling algorithms are frequently adopted due to their low time complexity and good results. Here, task nodes are ordered and selected in non-increasing order of their priorities (or ranks) and scheduled on the processors in order to optimize various performance metrics. Clustering is an efficient way to reduce communication delay in DAGs by grouping heavily communicating tasks to same labeled cluster and then assigning tasks in a cluster to the same resource. The *clustering based scheduling* algorithms are based on unbounded number of processors and therefore are not directly applicable. In *duplication based scheduling* algorithms, parents of current selected task are duplicated to reduce the task finish time.

List based Scheduling

In list scheduling, tasks are assigned with priorities, and placed in non-increasing order of priority list. Then, the tasks are considered for assignment to processors according to the order of priority list [27]. Some of the frequently used priority terms for a workflow task are: *s-level* (computation cost along the longest directed path from the concerned task to the exit task in the given DAG), *b-level* (same as *s-level*, but considering communication costs as well), *t-level* (computation and communication costs along the longest directed path from the entry task to the concerned task excluding its computation cost), etc.

The various list heuristics differ in how the priority is defined, and at what time a task is considered ready for assignment.

HEFT: Topcuoglu et al. [124] presented a list heuristic called *Heterogeneous Earliest Finish Time* (HEFT) algorithm. The HEFT algorithm selects the task with the highest b-level at each step. The selected task is then assigned to the processor that minimizes its earliest finish time with an insertion based approach which considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on the same resource. The time complexity of HEFT is $O(e \times p)$, where e is the number of edges in application DAG and p is the number of processors. Let us consider an example for which the HEFT produces a poor schedule. Figure 2.3(a) shows a simple DAG, and a computing system consisting of two processors connected with a single link [91]. Here, $comp(n_i, p_j)$, the computational cost of task n_i on processor p_j , is assumed to be 100 and $comm(e_{lm}, l_{12})$, the communication cost between tasks n_l and n_m , is set to 200. In this example, the communication cost is set high as large amount of data transfers through a WAN link. The example in Figure 2.3 illustrates that a schedule generated by HEFT may be a poor schedule [67].

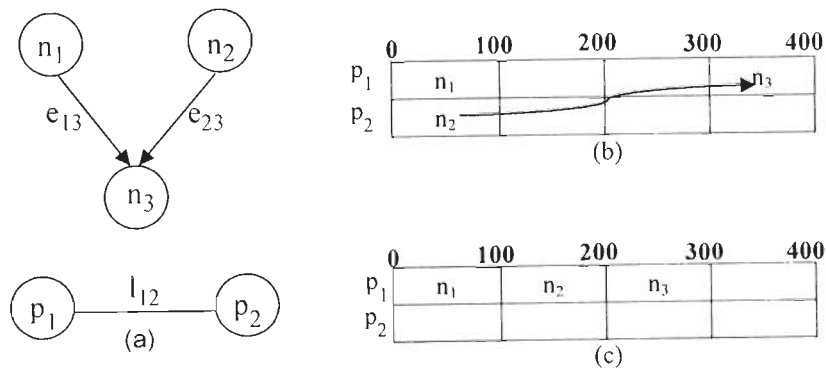


Figure 2.3: (a) DAG and System; (b) HEFT schedule; (c) A better schedule

FCP: Radulescu and Gemund [98] proposed a list heuristic called *Fast Critical Path* (FCP), intending to reduce the complexity of the list heuristics while maintaining the scheduling performance. Basically, a list heuristic has the

following procedures: the $O(e+n)$ time for ranking phase, the $O(n\log n)$ time for ordering phase, and finally the $O((e+n)\times p)$ time for resource selecting phase, where e is the number of edges, n is the number of tasks and p is the number of resources. Usually the third phase is larger than the second phase.

The FCP algorithm does not sort all the tasks at the beginning but maintains only a limited number of tasks sorted at any given time. Instead of considering all processors as possible targets for a given task, the choice is restricted to either the processor from which the last data arrives to the given task or the processor which becomes idle at the earliest. As a result, the time complexity is reduced to $O(n\log p + e)$.

LDCP: Daoud and Kharmia [29] presented a novel task scheduling algorithm, called the *Longest Dynamic Critical Path* (LDCP) for heterogeneous distributed computing systems (HeDCSs). The LDCP algorithm is a list based scheduling algorithm that uses a new attribute called Dynamic critical path (DCP) to effectively compute the priorities of tasks in HeDCSs. The DCP is simply a CP that considers the cancellation of communication costs among tasks scheduled in the same processor. At each scheduling step, the LDCP algorithm selects the task with the highest priority and assigns the selected task to the processor that minimizes its finish execution time using an insertion-based scheduling policy.

List Heuristics in Grid Computing: A list scheduling algorithm proposed in [136] is similar to the HEFT algorithm, but it modifies the method to compute the level of a task node by not only including its longest path to an exit node, but also taking incoming communication cost from its parents into account.

Ma and Buyya [84] proposed a new list heuristics called *Extended Dynamic Critical Path* (xDCP) which is a Grid enabled version of the Dynamic Critical Path (DCP) algorithm which was applied in homogenous computing environment. The idea behind DCP is continuously shortening the critical path

in the task graph by scheduling tasks in the current CP to a resource where a task on the critical path has an earlier start time.

The xDCP algorithm was proposed for scheduling parameter-sweep applications in a heterogeneous grid. The parameter sweep application (PSA) executes the same piece of code multiple times with unique sets of input parameters [128]. The improvements include: (i) initial shuffling of tasks to multiple resources when the scheduling begins instead of keeping them on one node, (ii) using the finish time instead of start time to rank task nodes to adapt heterogeneous resources, and (iii) multiple rounds of scheduling to improve the current scheduling instead of scheduling once. The complexity of xDCP is $O(n^3)$, which is same as that of DCP.

The drift towards new challenges in Grid computing implies the need for new, robust, multi-criteria scheduling algorithms. It includes execution time, the cost of running a task on a machine, reliability, and different data quality metrics. In [130], Wieczorek et al. propose a bi-criterion scheduling heuristic called Dynamic Constrained Algorithm (DCA) that uses novel requirement specification method based on a sliding constraint and model the problem as an extension of the multiple-choice knapsack problem. DCA outperforms existing algorithms designed for the same problem. It also shows relatively low scheduling times for workflows of medium size.

Duplication Based Scheduling

The concept behind duplication based heuristics is to utilize idle time of a resource to duplicate predecessor tasks. This may reduce the communication cost in transferring of results from a predecessor to a successor and thus, well suited for grid environments where communication cost can be very high between precedence-constrained tasks. Usually, duplication based algorithms have higher complexity than the algorithms discussed above. Most of the available duplication based algorithms are designed under the assumption of unbounded number of fully connected processors.

Duplication algorithms like the Critical Path based Full Duplication algorithm (CPFD) [5], Duplication First Reduction Next algorithm DFRN [92], Bottom up Top down Duplication Heuristic BTDH [27], Duplication Scheduling Heuristic DSH [69], algorithms proposed by Papadimitriou and Yannakakis [91], and Liou and Palis [82] try to duplicate all possible ancestors of a given node to improve performance, and in the process, become quite complex and resource consuming [5].

DPD: It is a heuristic strategy called the *Dominant Predecessor Duplication* (DPD) scheduling algorithm, which uses system heterogeneities and communication bandwidth to exploit the potential of parallel processing [73]. The algorithm can improve system utilization and avoid redundant resource consumption, resulting in better schedules. Tasks are scheduled according to system heterogeneity, network bandwidth, and communication requirements of applications. The objective of this approach is to duplicate dominant tasks (parents of candidate task who is responsible for late start of its successor on other processors) in order to reduce the candidate's start or finish time by decreasing the communication overhead.

HLD: Bansal et al. [9] proposed another duplication-based heuristics based on limited duplication, known as *Heterogeneous Limited Duplication* (HLD) for interconnection constrained networks to avoid the useless or redundant duplications in full duplication based algorithms. Here, if at any stage, it is found that duplication might increase the finish time of a task, then the duplication is not performed. The algorithm works efficiently with limited interconnection constrained multiprocessors and at high CCR (communication to computation ratio).

TANH: A task duplication-based algorithm called TANH (Task duplication-based scheduling algorithm for network of heterogeneous systems) is presented in [7, 102]. Compared to the version for homogeneous resources, the heterogeneous version has higher complexity, which is $O(n^2 \times p)$. This is

reasonable since the execution time of a task differs on different resources. A new parameter is introduced for each task: the *favorite processor* (fp), which can complete the task earliest. Other parameters of a task are computed based on the value of fp . In the clustering step, the initial task of a cluster is assigned to its first fp , and if the first fp has already been assigned, then to the second and so on. A processor reduction algorithm is used to merge clusters when the number of processors is less than the clusters generated [7].

The computational Grid usually has abundant computational and high communication cost resources (recall that the number of resources is unbounded in some duplication algorithms). This can make task duplication quite cost effective in grid environments. Duplication has already received some attention [114,117], but current duplication based scheduling algorithms in the Grid only deal with independent jobs. There are opportunities to create new algorithms for complicated DAGs scheduling in an environment that is not only heterogeneous, but also dynamic.

Clustering Based Scheduling

In parallel and distributed systems, clustering is an efficient way to reduce communication delay in DAGs by grouping heavily communicating tasks to the same labeled clusters, and then assigning tasks in a cluster to the same resource. In general, clustering algorithms have two phases: the *task clustering phase* that partitions the original task graph into clusters, and a *post clustering phase* which can refine the clusters produced in the previous phase and get the final task-to-resource map.

Clustering algorithms map tasks in a given DAG to the unlimited number of resources. In practice, an additional cluster merging step is needed after clusters are generated, so that the number of clusters generated can be equal to the number of processors. A task cluster could be linear or nonlinear. A clustering is called nonlinear if two independent tasks are mapped in the same

cluster; otherwise it is called linear. Figure 2.4 (a) shows a DAG with communication costs among tasks. The computation cost of tasks are $\{n_1 = 1, n_2 = 5, n_3 = 1, n_4 = 2, n_5 = 2, n_6 = 1, n_7 = 1\}$. Figure 2.4 (b) shows a linear clustering with three clusters $\{n_1, n_2, n_7\}$, $\{n_3, n_4, n_6\}$, and $\{n_5\}$ and Figure 2.4(c) presents a nonlinear clustering with clusters $\{n_1, n_2\}$, $\{n_3, n_4, n_5, n_6\}$, and $\{n_7\}$ [51]. The problem of obtaining an optimal clustering of a general task graph is NP-complete, so heuristics are designed to deal with this problem [51, 81, 82, 135].

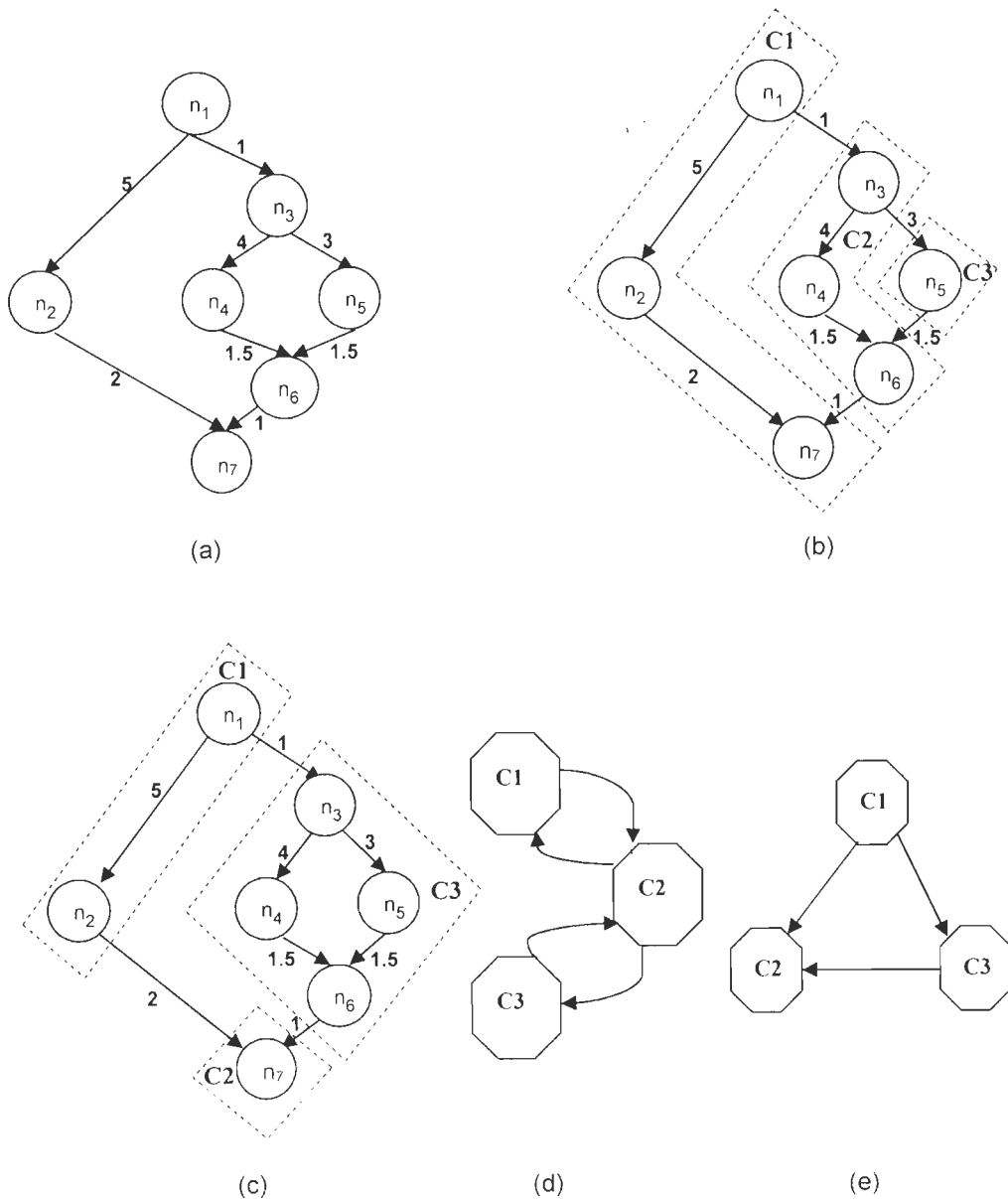


Figure 2.4: (a) DAG; (b) and (d) Linear clustering; (c) and (e) Non-linear clustering [51]

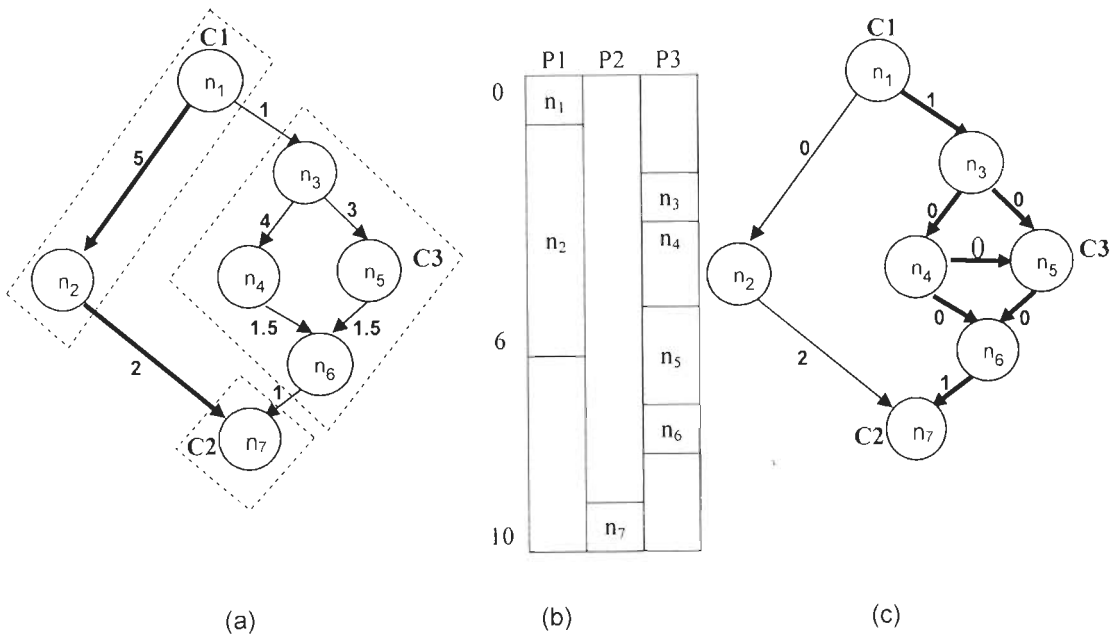


Figure 2.5: (a) A clustered DAG and its CP shown in thick arrows; (b) The Gantt chart of a schedule; (c) The scheduled DAG and the DS shown in thick arrows [51]

DSC: Yang and Gerasoulis [135] proposed a clustering heuristic called *Dominant Sequence Clustering* (DSC) algorithm. The critical path of a scheduled DAG is called *Dominant Sequence* (DS) to distinguish it from the critical path of a clustered DAG. The critical path of a clustered graph is the longest path in that graph, including both non-zero communication edge cost and task weights in that path. The makespan in executing a clustered DAG is determined by the Dominant Sequence, not by the critical path of the clustered DAG. Figure 2.5 (a) shows a critical path $\{n_1, n_2, n_7\}$ with a length of 9 of clustered graph. Figure 2.5(b) is showing a schedule of this clustered graph, and Figure 2.5 (c) gives the DS of the scheduled task graph, which consists of $\{n_1, n_3, n_4, n_5, n_6, n_7\}$ with a length of 10 [51].

In DSC algorithm, tasks priorities are dynamically computed as the sum of their t-level and b-level. The t-level and b-level are the sum of the computation

and communication costs along the longest path from the given task to an entry task and an exit task, respectively. While the b-level is statically computed at the beginning, the t-level is computed incrementally during the scheduling process.

Tasks are scheduled in the order of their priorities. The current node is an unassigned node with highest priority. Since the entry node always has the longest path to the exit node, clustering always begins with the entry node. The current node is merged with the cluster of one of its predecessors so that the top level value of this node can be minimized. If all possible merging increases the t-level value, the current node will remain in its own cluster. After the current node is clustered, priorities of all its successors will be updated.

The time complexity of DSC is $O((e+n)\log n)$, in which $O(\log n)$ comes from priority updating at each step using a binary heap, and $O(e+n)$ is for graph traversal in the clustering iterations. So for a dense task graph, the complexity is roughly $O(n^2 \log n)$.

2.1.3 Dynamism of Grid

There is an important issue for Grid computing which has not been discussed: *Resource Performance Dynamism*. All algorithms that we have discussed work under the assumption that the resource performance remains fixed, which may not necessarily be true in grid environment due to resource performance fluctuations during the execution. This problem could be reconciled by considering the possibility of conflict when a scheduling decision is made. He et al. [59] presented an example of this approach. Their algorithm considers the optimization of DAG makespan on multi-clusters which have their own local schedulers and queues shared by other background workloads, which arrive as a linear function of time.

The motivation is to map as many tasks as possible to the same cluster in order to fully utilize the parallel processing capability, and at the same time reduce the inter-cluster communication. The schedulers have a hierarchical structure: *Global Scheduler* is responsible for mapping tasks to different clusters according to their latest finish time in order to minimize the excess over the length of critical path. The *Local Scheduler* on each multi-cluster provides the estimated finish time of a particular task on this cluster, reports it to the Global Scheduler upon queries, and manages its local queue in a FIFO way. The time complexity of the global mapping algorithm is $O(p(n+1)n/2+e)$, where p is the number of multi-clusters.

Another approach to deal with dynamism is using dynamic algorithms. In [84], the authors propose a pM-S algorithm which extends a traditional dynamic Master-Slave scheduling model. In the pM-S algorithm, two queues are used by the master: the Unscheduled Queue, and the Ready Queue. The tasks in the ready queue can be directly dispatched to slave nodes, and a task in the unscheduled queue can be put into the ready queue when all of its parents have been in the ready queue or dispatched. The dispatching order in the ready queue is based on tasks' priorities. When a task is finished, the priorities of all its children's ancestors will be dynamically promoted.

In [63], another dynamic algorithm is proposed for scheduling DAGs in a shared heterogeneous distributed system. Unlike the previous works in which a unique global scheduler exists, in this work, the authors consider multiple independent schedulers which have no knowledge about other jobs. So there is the danger of conflicts on resources. This algorithm is derived from a static list algorithm: the *Dynamic Level Scheduling* (DLS). In the original DLS, the dynamic level of a task in a DAG is used to adapt to the heterogeneity in resources, while in the newly proposed algorithm, the dynamic length of the queue on each resource is also taken into account for computing a task's level. To estimate the length of the queue, it is assumed that jobs are coming following a Poisson distribution. The research also discusses how to choose the

time when the scheduling decision is made and the time when a task should be put into resource local queue in a dynamic system. Ready task queuing probabilities, which are computed following the Poisson distribution, are used to make these decisions.

All DAG scheduling algorithms that have been discussed so far have the same goal: minimizing the makespan of a task graph. In [140], Yu et al. consider the scheduling problem in a “pay-per-use” service grid. Their objective is to minimize the total cost for executing a workflow on such a Grid, and at the same time QoS, which is interpreted as a deadline, is provided. The algorithm firstly partitions the original DAG into sub-workflows, which consist of a sequential set of tasks between two synchronized tasks (the nodes from which at least one sub-workflow starts and/or ends) in the graph, and assigns a sub-deadline to each partition by a combinational *Breadth First Search* and *Depth First Search* with critical path analysis. 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. Rescheduling is also provided when a sub-workflow misses its sub-deadline. But only the sub-workflow’s children will be rescheduled to reduce the rescheduling cost.

A similar problem is also considered by Sample et al. [109]. They adopted similar scheduling approach as used by Yu et al. [140]. The scheduling begins with the selection of an initial schedule based on service providers’ estimates for completion time and cost. 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. To find an initial scheduling, the scheduler requests bids from resource providers for the services they can provide. The bid request is based on the service needed, the expected start time for the service, and information about the size and complexity of the input parameters to the service.

The uncertainty introduced by the dynamism also brings opportunities for application of a new approach in the Grid: the *Data and Control dependency task Graph* (DCG) scheduling [35]. Different to a DAG, a DCG has two types of edges denoting data dependency and control dependency among task nodes. A control-dependency edge usually represents some condition relying on the results of its starting node, so in a DCG, users can predefine adaptive rules for dynamic adaptation in their task graph. For example, a rule can be duplicating all the successors of a node if its real execution time fails to match its predicted one. The research [35] shows an example of how to schedule such conditional task graphs in terms of reducing the final possible makespan. However, no research, that adopts this idea in the grid environment, exists at this time.

2.1.4 Dynamic Rescheduling

When there is no resource prediction service available or the resource prediction cannot provide an accurate forecast, rescheduling which changes previous schedule taking current resource into account can bring some improvement. With the efforts of system designers and developers, more and more grid infrastructures now support job migration, which is one precondition of rescheduling [74, 126].

In [133], a self-adaptive scheduling algorithm is given to reschedule tasks on the processors showing “abnormal” performance. Abnormal performance means that the behaviors of those processors violate the performance model with an unexpected high local job arrival rate that delays remote grid jobs. The algorithm uses a prediction error threshold to trigger the rescheduling process. If the estimated completion time of a grid task is shortened after migration, the task will be migrated to the processor which gives minimum completion time according to current prediction. The key problem in this algorithm is to find a proper threshold to determine whether a processor is

abnormal. The problem with this algorithm was that it ignored the migration overheads when it computed the benefits of migrations.

The above methods only consider the rescheduling of independent tasks in the Grid. Sakellariou and Zhao [107] proposed a novel low cost rescheduling policy to improve the initial static schedule of a DAG. This algorithm only considered a selective set of tasks for rescheduling based on measurable properties. The key idea of this selective rescheduling policy is to evaluate (at run-time, before each task starts execution) the starting time of each node against its estimated starting time in the static scheduling and the maximum allowable delay in order to make a decision for rescheduling.

The maximum allowable delay could be the slack of a task, or the minimal spare time. The slack of a task is defined as the maximal value that can be added to the execution time of a task without affecting the overall makespan of the schedule. The minimal spare time is the maximum value that can be added to the execution time of a task without delaying its successors' execution. As the tasks of the DAG are running, the re-scheduler maintains two schedules: One for the static scheduling using estimated values, and the other for keeping track of the status of the tasks executed so that the gap between the original schedule and the real run can be known. Once the gap is beyond the slack or the minimal spare time, rescheduling of all unexecuted tasks will be triggered and a new slack or minimal spare time for each task is computed. This rescheduling policy is applied to some list heuristics discussed in Section 2.1.2 such as HEFT, FCP, etc. Simulation results show that this policy can achieve comparable performance with the policy that reschedules all tasks in the DAG while dramatically reducing the rescheduling overhead.

Lee et al. [76] proposed a Dynamic HEFT (DH) algorithm to enhance the functions of the original static HEFT (SH) algorithm. Instead of dispatching tasks to physical processors directly, the DH algorithm dispatches tasks to multiple queues. During runtime, the DH algorithm continues to dispatch the

scheduled tasks to corresponding physical processor and predicts the transfer rate. Once the difference between two consecutive transfer rates is greater than the threshold value, rescheduling will be performed. The experimental results show that the proposed DH algorithm performs better than the SH algorithm, especially under the Grid computing environment with fluctuant transfer rates and high bandwidth differences.

2.1.5 Nature's Laws Inspired Scheduling

The nature's heuristics were only relatively introduced into scheduling area and more research needs to be done to fit them in the grid context. Several analogies from natural phenomena have been introduced to form powerful heuristics, which have proven to be highly successful. Abraham et al. [3] and Braun et al. [19] presented three basic meta-heuristics implied by nature for grid scheduling, namely, *Genetic Algorithm (GA)*, *Simulated Annealing (SA)*, and *Tabu Search (TS)*, and heuristics derived by a combination of these three algorithms. Meta-heuristics are high level heuristics that guide local search heuristics to escape from local optima.

Genetic Algorithm (GA): Genetic algorithms are most widely studied guided random search techniques for task scheduling [30]. It executes in generations, producing better and better solutions using crossover and mutation operators in each generation and randomly producing new solutions (offspring) for the next generation based on the solutions (parents) in the current solution [62, 132]. In a genetic algorithm, historical knowledge can be used to guide the chromosome selection, crossover, or mutation process so that the search process can converge quickly. GA produces good quality of schedules but their execution times are significantly higher than the other. It is found that GA based heuristics require around a minute to produce a solution, while the other heuristics require an execution of few seconds.

Simulated Annealing (SA): SA is a general random search technique based on the physical process of annealing involving repeated heating and cooling [19, 68]. By analogy, the thermal equilibrium is an optimal task–machine mapping (optimization goal), the temperature is the total completion time of a mapping (cost function), and the change of temperature is the process of mapping change. If the next temperature is higher, which means a worse mapping, the next state is accepted with certain probability. This is because the acceptance of some “worse” states provides a way to escape local optimality which occurs often in local search. In SA, the cooling rate is usually set to a fixed value to control the number of iterations the search process will perform. Here the question is “How to select this value as the threshold value for accepting a poorer mutation?” Second, there is a trade–off between the search cost and the degree of optimality of solutions found.

Tabu Search (TS): TS is a method of keeping track of the regions of the solution space that have already been searched in order to avoid repeating the search near these areas [19, 32]. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a successful long hop). The intuitive purpose of a long hop is to move to a new region of the solution space that has not already been searched. After each successful long hop, the short hop procedure is repeated. The stopping criterion for the entire heuristic is when the sum of the total number of successful short hops and successful long hops equals limit hops. Then, the best mapping from the tabu list will be the final solution.

Combined Heuristics: GA can be combined with SA and TS to create combinational heuristics. For example, The *Genetic Simulated Annealing* (GSA) heuristic is a combination of the GA and SA techniques [3]. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule, system temperature, and a simplified SA decision process for accepting or rejecting a new chromosome.

There exist few algorithms which show how QoS requirements affect the resource assignment in the Grids. The grid resource broker does not own the control of local schedules which introduces the load imbalance over the resources. Hence, a good policy is needed to uniformly distribute the tasks to resources and maintain the work load over resources.

2.2 Research Directions

From literature survey, it was seen that researchers have worked on two types of applications, namely independent task scheduling and workflow applications modeled as DAGs. This research work is also focusing on these two types of applications. Some observations from the survey which guided our thought process are discussed below.

It has been observed (Section 2.1.1) that OLB algorithm is well suited as far as load balancing is concerned for independent task scheduling. Also, Max–Min and XSuffrage produce good schedules for independent tasks. It would be expected that these algorithms may work well in heterogeneous grid environments also. In case of workflow applications, list scheduling heuristics are widely adopted due to their easy implementation, but they produce less efficient schedules. Duplication based algorithms have opportunity to improve list based schedules though with higher complexity and resource consumption. Thus, duplication strategy can be useful in scheduling workflow application in grids if the schedules generated can be further optimized to reduce the resource consumption. The matter of higher complexity in duplication based scheduling heuristics can be tolerated due to the static nature of tasks and these tasks are submitted offline to the Grid. In case of online job submission, duplication may not be very useful. But, duplication can be a useful strategy for static workflow scheduling in grids.

Further, single objective scheduling techniques cannot fulfill all the demands of grid scheduling, and at times, the different objectives cannot be improved simultaneously due to conflicting requirements. Resource owners in grid are interested in maximizing resource utilization and economic profit, whereas resource consumers are aiming to minimize the execution time and economic cost for running their applications. This multi-objective nature of scheduling in grids motivates us to develop new multi-objective scheduling algorithms for grid computing systems. A multi-objective scenario can be minimization of execution time and economic cost of application tasks, minimization of processor consumption of grid resources, maximization of trustworthiness/reliability of grid resources, and maximization of QoS requirements of tasks.

The research efforts in this thesis are guided by the above gaps or directions that have been identified through literature study. Specifically, our research work addresses the following:

- (a) Modeling of resource (processor and network) heterogeneity in grid for workflow scheduling.
- (b) Improving existing duplication-based scheduling algorithms for workflow applications in heterogeneous grid environment.
- (c) Development of bi-criteria scheduling algorithms for workflows (DAGs) while exploiting duplication to optimize schedule length without letting it increase the economic cost.
- (d) Development of multi criteria scheduling algorithms for optimizing execution time and economic cost while meeting QoS requirements such as trustworthiness, resource availability, and communication bandwidth.

3.1 Overview

In Grid scheduling, a variety of optimization criteria are of interest e.g., minimization of total execution time (makespan), minimization of total economic cost as per resource usage, maximization of resource availability, maximization of resource trustworthiness and application reliability, etc. The recent research trends reveal that execution time, economic cost, trustworthiness, and reliability are the primary objectives of researchers to design more efficient scheduling techniques to build robust and reliable distributed computing environment. In this chapter, we have investigated new heuristics for time-based and cost-based scheduling objectives. In time-based scheduling, heterogeneity of resources has been focused and schedule length has been optimized with an effective scheduling strategy. In cost-based scheduling, schedule cost or processor consumption is an objective in which new duplication-based heuristics have been proposed for heterogeneous and homogeneous environments in order to minimize processor consumption.

3.2 Time Based Scheduling

3.2.1 Preamble

Heterogeneity is not new to scheduling algorithms but it is a big challenge for scheduling tasks in grid computing environments. In Grid computing, resources (both computational and storage) are distributed in multiple domains, and the underlying networks connecting them are heterogeneous. The heterogeneity reflects the different capabilities for task execution and data access. A study [85] shows that if high performance is required in a computational grid, the scheduler should have the ability to adapt different application/resource heterogeneity. Thus, heterogeneity of tasks/resources cannot be neglected while scheduling an application in Grid computing environment. In this chapter, a list-based task scheduling algorithm, known as *Scheduling with Heterogeneity using Critical Path* (SHCP) has been presented and discussed. The objective of a task scheduling problem is to map (match and schedule) tasks onto the suitable resources, and to order their execution on each resource such that precedence relationships between tasks are not violated and the overall execution time (makespan) could be minimized.

The heterogeneous resources (e.g., processors, data storages, catalogs, network resources, and sensors) in the grid are spread over large geographical region connected through arbitrary topology. It causes challenges for scheduling applications because exact estimation of computation and communication costs, at times, may not be possible due to its dependence on multi-variable parameters and on run-time input data. Although, dynamic scheduling algorithms [76] are more realistic as they consider the actual costs at run time and may offer a better load balancing and system utilization. The static compile time task scheduling algorithm (SHCP) presented here does not incur the run time scheduling overheads as is the case with the dynamic scheduling techniques.

Bote–Lorenzo et al. [17] pointed out that a grid system has the characteristics of heterogeneity, large scale and geographical distribution. It has been observed that there exists a high level of heterogeneity among internet routers, peer-to-peer and grid systems. Scheduling precedence constrained task graphs (DAGs) becomes more complex in grid environment and the increasing heterogeneity may be quite detrimental if not handled properly. Zhao and Sakellariou [143] showed that length of the schedule produced may be affected significantly with appropriate selection of task and edge weight in the DAG using mean, median, best or worst values.

In literature [9,106, 124, 143], researchers have considered the task node value in heterogeneous system as an average, median, best or worst value etc. Most of the research work done on scheduling DAG in distributed heterogeneous environment is based on list based heuristics like Heterogeneous Earliest Finish Time (HEFT) [124], Critical Path on Processor (CPOP) [124] and Hybrid Heuristic Scheduling (HHS) [106]. Zhang et al. [142] presented a relative performance comparison of scheduling algorithms in the Grid environment. It shows that HEFT and HHS algorithms perform better than level based scheduling methods on many combination of computing environments and DAGs in the Grid [37].

The HEFT algorithm is an insertion based static list scheduling heuristic that assigns priorities of task nodes on the basis of higher b–level. The highest priority task is scheduled on the processor which finishes its execution at the earliest. Scheduling holes are also exploited if needed to improve the start time of a task. The HHS is a class of algorithms that uses hybrid version of the list based and level–based scheduling [37] approaches. It partitions the DAG into different levels of independent tasks. The tasks in each level are ordered and scheduled as used by HEFT. In this thesis, a new approach for computing the priority of task nodes is adopted considering heterogeneity of computing resources and underlying networks which have not been considered in the literature till now. The priorities of task nodes decide the execution order of

tasks which reflects the schedule length of task graph. The experimental results show that SHCP algorithm performs better for running large task graphs (workflows) in Grid based heterogeneous environment at high CCRs. A high CCR (communication to computation ratio) indicates the communication intensive nature of a problem, whereas, low CCR represents the computation intensive problem.

3.2.2 Grid Resource Model

A Grid resource model (GRM) has been considered where heterogeneous resources are connected through arbitrary topology and network bandwidth varies from link to link. A Grid resource model can be represented by $G = (P, Q, A, B)$, where $P = \{p_i \mid p_i \in P \text{ for } i = 1, 2, \dots, p\}$ is the set of p highly available bounded number of computing resources, $A = \{\alpha(p_i) \mid \alpha(p_i) \in A \text{ for } i = 1, 2, \dots, p\}$ is the set of processing rates, where $\alpha(p_i)$ is the processing rate of resource p_i (i.e., the unit is instruction counts/time, such as one million instructions/second), $Q = \{q(p_i, p_j) \mid q(p_i, p_j) \in Q \text{ for } i, j = 1, 2, \dots, p\}$ is the set of communication links connecting pairs of distinct resources, $q(p_i, p_j)$ is the communication link between p_i and p_j , and $B = \{\beta(p_i, p_j) \mid \beta(p_i, p_j) \in B \text{ for } i, j = 1, 2, \dots, p\}$ is the set of data transfer rates, where $\beta(p_i, p_j)$ is the data transfer rate between p_i and p_j (i.e., the unit is volume/time, such as Kbytes/second).

In this model, it is assumed that each processor has co-processor to deal with communications, which allows computation and communication to overlap each other. Additionally, task executions are assumed to be non-preemptive and intra-processor communication cost between two tasks scheduled over the same resource is considered as zero.

3.2.3 Workflow Application Model

A workflow application model (WAM) may be represented by a DAG, $W=(N,E,T,C)$ as shown in Figure 1.1, where N is a set of n computation tasks, T is a set of task computation volumes (i.e., one unit of computation volume is one million instructions), E is a set of communication edges that shows precedence constraint among the tasks and C is the set of communication volumes (i.e., one unit of communication volume is one Kbyte). The value of $\tau_i \in T$ is the computation volume for task $n_i \in N$. The value of $c_{ij} \in C$ is the communication volume occurring along the edge e_{ij} , where $e_{ij} \in E$ is an edge between task n_i and n_j for $n_i, n_j \in N$. A task node without any parent node is called entry node, and a task node without any child node is called exit node.

3.2.4 Matter of Heterogeneity in Grids

In SHCP, the expected execution cost of tasks and expected communication costs between the different pairs of tasks are computed using processor and network heterogeneity factors formulated by us. In this model, a priority based task sequence is generated using critical path. A *Critical Path (CP)* can be defined as the longest directed path in terms of computation and communication costs in the DAG in depth first search manner. In SHCP, an unscheduled task is selected from the CP-based task sequence and scheduled on the processor (or resource) that minimizes its earliest finish time with an insertion-based approach. The comparison study shows that SHCP algorithm performs better in terms of performance metrics in Grid based heterogeneous environment for large task graphs of different sizes and at high CCRs.

Heterogeneity is a type of variability in characteristics (execution rate, communication bandwidth, etc) of the resources (computational, storage,

network, etc) of any distributed computing system [17]. In this section, a heterogeneity model is considered which defines two terminologies (processor heterogeneity factor and network heterogeneity factor) to estimate the expected computation costs of tasks and expected communication costs of edges in the given DAG [96]. The *processor heterogeneity factor* ρ can be computed as:

$$\rho = \frac{2 \times \sqrt{\frac{\sum_{i=1}^p (\alpha(p_i) - \overline{\alpha(p_i)})^2}{p}}}{\max\{\alpha(p_i)\}} \quad (3.1)$$

where $\overline{\alpha(p_i)}$ is the mean processing rate. It can be computed as:

$$\overline{\alpha(p_i)} = \frac{\sum_{i=1}^p \alpha(p_i)}{p} \quad (3.2)$$

A high value of α shows the high heterogeneity among the resources. Similarly, the *network heterogeneity factor* σ can be computed as:

$$\sigma = \frac{2 \times \sqrt{\frac{\sum_{i=1}^p \sum_{j=i+1}^p (\beta(p_i, p_j) - \overline{\beta(p_i, p_j)})^2}{(p^2 - p)/2}}}{\max\{\beta(p_i, p_j)\}} \quad (3.3)$$

where $\overline{\beta(p_i, p_j)}$ is the mean transfer rate. It can be computed as:

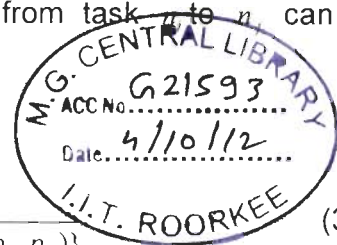
$$\overline{\beta(p_i, p_j)} = \frac{\sum_{i=1}^p \sum_{j=i+1}^p \beta(p_i, p_j)}{(p^2 - p)/2} \quad (3.4)$$

It is assumed that data transfer rate between two computing resources without any direct link is zero. Thus, the *expected computation cost* of task nodes can be computed as:

$$\tilde{\omega}_i = \frac{\tau_i}{\rho \times \min\{\alpha(p_i)\} + (1 - \rho) \times \max\{\alpha(p_i)\}} \quad (3.5)$$

Similarly, *expected communication cost* of edge from task p_i to n_i can be computed as:

$$\tilde{\varepsilon}_{ij} = \frac{c_{ij}}{\sigma \times \min\{\beta(p_i, p_j)\} + (1 - \sigma) \times \max\{\beta(p_i, p_j)\}} \quad (3.6)$$



3.2.5 Heterogeneity Aware Critical Path Based Scheduling

The whole process of task scheduling onto the Grid resources can be divided in two phases: (a) Critical Path Based Task Sequence Generation, and (b) Resource Selection and Task Assignment.

(a) Critical Path Based Task Sequence Generation

In this phase, a critical path based task sequence is generated using expected computation and communication costs obtained from Equations (3.5) and (3.6). The nodes along this critical path are abbreviated as CP nodes. After constituting the critical path, other nodes are added to keep the precedence constraint order of task execution. The predecessor nodes are added on the basis of higher b-level, and ties being broken on the basis of lower t-level [9]. Other remaining nodes are added using the same priority as assigned to them at the end of task sequence. The b-level of task n_i is the longest directed path

considering computation cost and communication cost from n_i to the exit node in the DAG. It can be computed recursively using the following formula:

$$b_i = \tilde{\omega}_i + \max\{b_j + \tilde{\varepsilon}_{ij}\} \quad \forall n_j \in \text{succ}(n_i) \quad (3.7)$$

where $\text{succ}(n_i)$ refers to the immediate successors of task node n_i in the DAG. Similarly, the t-level of n_i is the longest directed path considering computation cost and communication cost from node n_i to entry node in the DAG excluding its computation cost. It can be calculated as follows:

$$t_i = \max\{t_j + \tilde{\varepsilon}_{ij}\} \quad \forall n_j \in \text{pred}(n_i) \quad (3.8)$$

The first unscheduled task in the task sequence is known as candidate task.

(b) Resource Selection and Task Assignment

In this phase, the unscheduled selected candidate task is mapped to the processor in the processor network which allows it to finish at the earliest using insertion based approach. The task n_i can start its execution on the candidate processor if and only if data arrives from all of its immediate parents so as to meet the precedence constraints. To select the best processor for the candidate task, it is necessary to define the earliest start time (est) and earliest finish time (eft) of task n_i on processor p_j . The est of entry task node n_{entry} on processor p_j can be calculated as:

$$est(n_{entry}, p_j) = 0 \quad (3.9)$$

Similarly, eft and est must be computed recursively for the remaining tasks. To compute the eft of task n_i , all immediate predecessors of task n_i must have been scheduled.

$$est(n_i, p_j) = \max\{p_j^R, \max_{n_m \in pred(n_i)} \{aft(n_m) + \frac{c_{m_i}}{\beta(p_k, p_j)}\}\} \quad (3.10)$$

$$eft(n_i, p_j) = \frac{\tau_i}{\alpha(p_j)} + est(n_i, p_j) \quad (3.11)$$

where $aft(n_m)$ is the actual finish time of task n_m (scheduled on processor p_k) which is equal to earliest finish time $eft(n_m)$ of task n_m after successfully assignment on processor p_k . Also, p_j^R is the time when processor p_j is ready to execute new task in non-insertion based scheduling policy. After assignment of all tasks in a DAG, the makespan of the schedule will be the actual finish time of the exit task n_{exit} which can be computed as:

$$makespan = \max\{aft(n_{exit})\} \quad (3.12)$$

The pseudo code of the SHCP algorithm is presented in Figure 3.1. The SHCP works in two phases: (1) CP based task sequence generation phase for computing the priorities of tasks using processor and network heterogeneity factors, and (2) Task scheduling phase for selecting the tasks in order of their priorities and schedule on to a resource which minimizes the task's finish time. This algorithm critically evaluates the average computation and communication costs for generating the task sequence. This order of tasks in this sequence differs from task sequence obtained using HEFT [124] which affects the workflow schedule and consequently the schedule length of application.

Algorithm 3.1: SHCP

1. Compute *computation and communication heterogeneity factors* using Equations (3.1) and (3.3).
2. Set the weights of task nodes in DAG with the *expected computation cost* using Equation (3.5).
3. Set the weights of communication edges with *expected communication cost* using Equation (3.6).
4. Construct a *critical path based task sequence*.
5. **while** there are unscheduled tasks in task sequence ($n_i \in N$) **do**
6. Select the first unscheduled task n_i from the task sequence
7. **for** each processor p_j , ($p_j \in P$) **do**
8. Compute $eft(n_i, p_j)$ with insertion based task scheduling using Equation (3.11).
9. Assign task n_i to processor p_j which minimizes eft of task n_i .
10. **end while**

Figure 3.1: SHCP scheduling algorithm

3.2.6 Performance Comparisons and Result Analysis

We have implemented the scheduling algorithms (SHCP, HEFT and HHS) in the simulated Grid environment and tested with random DAGs to generate schedules produced on to the Grid of heterogeneous resources. The performance of SHCP is compared with the well known HEFT and HHS algorithms using the following performance metrics:

Schedule Length Ratio (SLR): Since a large set of task graph with different properties is used, it becomes necessary to normalize the schedule length (makespan) to a lower bound, called the Schedule Length Ratio (SLR) which can be computed as:

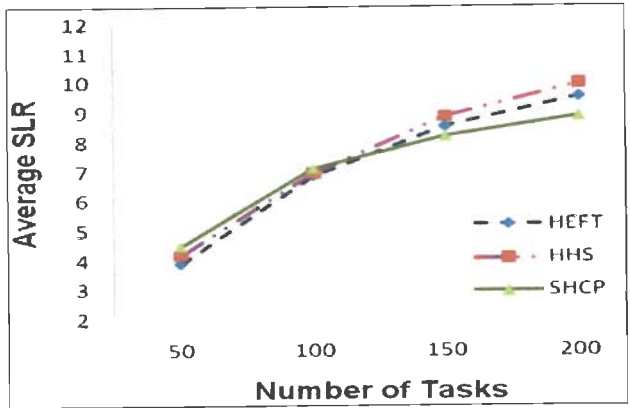
$$SLR = \frac{makespan}{\sum_{n_i \in CP_{\min}} \min_{p_j \in P} \left\{ \frac{\tau_i}{\alpha(p_j)} \right\}} \quad (3.13)$$

The denominator is the summation of the minimum execution costs of tasks on the critical path CP_{\min} as discussed in [124]. The average values of SLR over several random task graphs are used in the simulation.

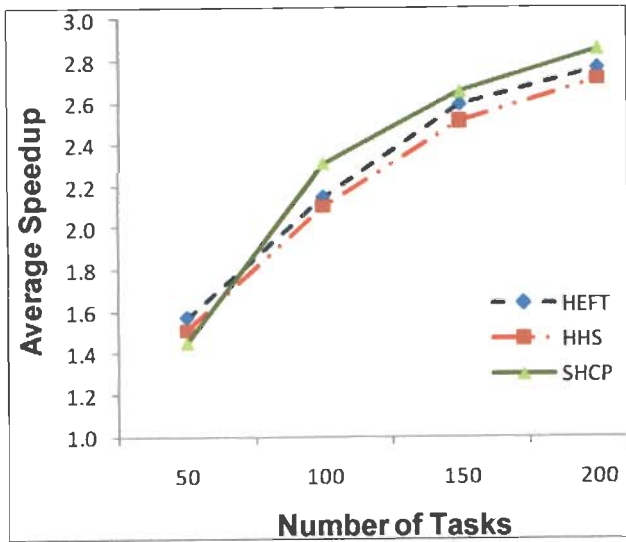
Speedup: It can be computed by dividing the minimal sequential execution time (cumulative computation costs of tasks on processor p_j that minimizes it) with the parallel execution time (makespan) which can be defined as:

$$Speedup = \frac{\min_{p_j \in P} \left\{ \sum_{n_i \in N} \frac{\tau_i}{\alpha(p_j)} \right\}}{makespan} \quad (3.14)$$

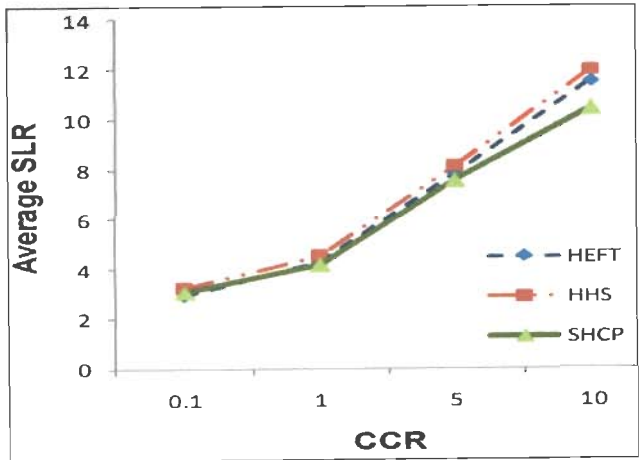
The results are obtained with respect to average SLR and average speedup over different workflows of various sizes and CCRs. Each result, obtained with respect to workflow size (i.e., number of tasks in a workflow) and CCR, is an average of 100 different task graphs. Different DAGs of order 50 to 200 are randomly generated in the simulation. The out degree of each task varies from 2 to 5. The CCR for different DAG applications varies from 0.1 to 10. The experimental results (Figure 3.2) show that SHCP algorithm surpasses the HEFT and HHS algorithms for large task graphs and at high CCRs in the simulated Grid computing environment. For small DAG applications, SHCP show ineffectiveness as effect of heterogeneity is very less to divert the task sequence and generated schedules become slightly worse while in case of large DAG applications, SHCP exceeds the HEFT and HHS. Figures 3.2 (a to c) show the improvement of our strategy with the earlier list based scheduling heuristics.



(a)



(b)



(c)

Figure 3.2: Performance comparison of SHCP algorithm

The performance of schedules depends on the estimation of computation costs of tasks and communication costs of data transfer between tasks. The SHCP algorithm considers the heterogeneity of nodes and edges in application DAG and compute the computation and communication heterogeneity factors which are helpful in building a better task sequence based on b-level. The order of tasks in a task sequence ultimately affects the schedule length. The formulation of heterogeneity factors in Equations (3.1) and (3.3) is based on 'Standard Deviation' which reflects the spread of execution times of a task in workflow. This is helpful in more accurate prediction of expected computation and communication costs at compile time in the grid environment.

The experimental results show that the proposed single criterion scheduling algorithm (SHCP) improves with the increase of workflow sizes and CCRs. In general, CCR is high in grid computing environment. The schedules produced using SHCP for large task graphs at higher CCRs are shorter than HEFT and HHS in Grids. SHCP minimizes the execution cost (makespan) of large sized workflow applications (communication-intensive) and hence is better suited to be adopted for grid computing environment.

3.3 Cost Based Scheduling

3.3.1 Duplication Cost Based Scheduling

In current economic market, users have to pay execution cost to run their applications over the grid resources which is computed as per the total run time consumed by them. Task duplication based scheduling algorithms generate shorter schedules without sacrificing efficiency but leave the computing resources over consumed due to heavy duplications which increases the execution cost [5, 8, 9, 37, 92]. We have presented two efficient duplication based scheduling algorithms (1) *Heterogeneous Economical Duplication* (HED) for heterogeneous (e.g., Grids), and (2) *Reduced Duplication* (RD) for

homogeneous (e.g., Clusters) computing systems. These algorithms reduce the processor/resource consumption without affecting the overall schedule length/makespan.

Duplication heuristics are more effective for fine grain task graphs and for networks with high communication latencies. Duplication plays its role more effectively at higher CCRs, as the formation of large sized scheduling holes increases with higher communication costs, which can be exploited to accommodate fine grain tasks conveniently [8]. The primary objectives of these algorithms (HED and RD) are to minimize duplications after duplicating tasks on processors selectively and minimize the overall schedule cost (Processor Consumption). Processor Consumption (PC_i) of a processor p_i is defined as the fraction of its total time involvement for an application that the processor is actually busy executing some tasks of this application. Overall percentage processor consumption (PC) of a schedule is then calculated as the average of the individual processor consumption.

$$PC = \frac{\sum_{i=1}^p PC_i}{\text{Number of processor booked}} \times 100 \quad (3.15)$$

In homogeneous computing systems, selective duplication (SD) scheduling algorithm duplicates the parent tasks selectively to start the execution of dependent tasks earliest, which results in lower duplications and lower time and space complexity [8]. Inter-process communication imposes performance limitations on scheduling algorithms. Duplication of tasks on more than one processor reduces the waiting time of dependent tasks. Most of the duplication-based scheduling algorithms try to duplicate all possible predecessors of a given task to improve the performance [5, 27, 69, 91, 92].

In heterogeneous computing systems, heterogeneity of computational resources and communication mechanisms poses some major obstacles to

achieve high parallel efficiency. The performance of the scheduling algorithms tends to degrade with an increase in heterogeneity and CCR which results in an inappropriate task/processor selection. In this case, duplication is very graceful to overcome these ‘stresses’ and ‘strains’ of heterogeneity by replicating the crucial tasks and thereby improving the finish time on processor, but it increases scheduling cost due to overheads of duplicated tasks.

Savina et al. [9] suggested a heterogeneous limited duplication (HLD) scheduling algorithm that adapts the SD algorithm [8] in heterogeneous environment, and then assessed the usefulness of limited duplication approach in dealing with the stresses of heterogeneity in a system. Dogan and Ozguner [37] proposed a level sorting algorithm (LDBS) to arrange the tasks in DAG into various precedence levels. The tasks, belonging to the same level have no data dependencies, can be executed concurrently. In LDBS, tasks are scheduled level by level starting from the top.

3.3.2 Resource Model

A target computing system can be represented by $M = (P, Q)$, where $P = \{p_i | p_i \in P \text{ for } i = 1, 2, \dots, p\}$ is the set of p fully connected bounded processors, $Q = \{q(p_i, p_j) | q(p_i, p_j) \in Q \text{ for } i, j = 1, 2, \dots, p\}$ is the set of communication links where $q(p_i, p_j)$ is the communication link from p_i to p_j . In heterogeneous computing system, task execution cost on different processors may be different due to the processor heterogeneity (different processing rates) and similarly, the data transfer rates (bandwidths) between different pair of processors may be different due to network heterogeneity (see Section 3.2.4). In this model, it is assumed that each processor has co-processor to deal with communications, which allows computation and communication to overlap each other. Additionally, task executions are assumed to be non-preemptive and communication overhead between two tasks scheduled on the same processor

is considered as zero. After completing execution of a task, the associated processor sends output data to all of its child tasks in parallel.

Heterogeneous processing is an abstract model with wide manifestations. Savina et al. [9] have categorized the heterogeneity models into two broad categories (1) Mixed heterogeneity machine model (MHM Model) and (2) Fixed heterogeneity machine model (FHM Model). In MHM model, the target machine consists of a mixed suite of p processors that are best suited to process a particular type of program code. Therefore, execution time of a task n_i on processor p_k will depend on, how well the architecture of p_k matches n_i 's processing requirements. A task scheduled on its best-suited processor will take lesser execution time, as compared to the one scheduled on a less-suited processor. It is to be noted that the best processor for one task may be the worst processor (one that takes maximum execution time/or does not execute at all the particular code type task) for the other. This type of model represents the heterogeneous machine suite as described by Khokhar et al. [66] and covers the type of heterogeneity modeled in [99, 101, 124], corresponding to a task having different execution times on different processors.

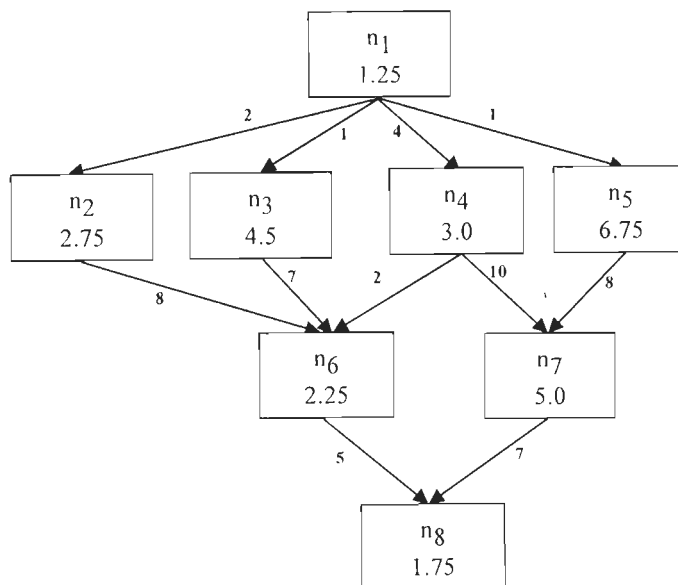


Figure 3.3: A simple DAG with precedence constraints

A homogeneous computing system is a special case of a heterogeneous computing system, in which all processors compute at the same execution rate and the communication overhead between any two processors is the also same. Figure 3.3 shows a DAG for workflow application model (see Section 3.2.3). The mean computation cost $\bar{\omega}_i$ of task n_i and mean communication cost $\bar{\varepsilon}_{ij}$ between task n_i and task n_j can be calculated as:

$$\bar{\omega}_i = \frac{\sum_{j=1}^p \omega_{ij}}{p} \quad \forall 1 \leq i \leq n \quad (3.16)$$

$$\bar{\varepsilon}_{ij} = \frac{c_{ij}}{\text{mean data transfer rate over all links in processor network}} \quad \forall 1 \leq i \neq j \leq n \quad (3.17)$$

where ω_{ij} is computation cost of task n_i on processor p_j . Figure 3.3 is showing a DAG where nodes represent mean computation costs ($\bar{\omega}_i$) of task nodes and edges represent the mean communication costs ($\bar{\varepsilon}_{ij}$) between task nodes.

Table 3.1: Computation cost matrix [ω_{ij}] for DAG in Fig. 3.3

Task Node	Computation costs on different processors, ω_{ij}				Mean cost, $\bar{\omega}_i$
	P1	P2	P3	P4	
n ₁	1	1	2	1	1.25
n ₂	3	2	4	2	2.75
n ₃	5	6	3	4	4.5
n ₄	2	4	4	2	3.0
n ₅	4	8	7	8	6.75
n ₆	3	3	1	2	2.25
n ₇	5	5	5	5	5.0
n ₈	1	2	2	2	1.75

3.3.3 Reducing Duplications – Needs and Approaches

This section presents the economical scheduling algorithms (HED and RD) based on duplication approach on a bounded number of fully connected processors in heterogeneous and homogeneous computing systems respectively. The HED and RD algorithm include two mechanisms, first is a lower-bound complexity mechanism for scheduling based on insertion-based task duplication and second is modifying schedule after removing some duplicated and unproductive tasks from the schedule without affecting the makespan.

In HED (Heterogeneous Economical Duplication), a priority based task sequence is generated by ordering the tasks in non-increasing order of their b-level. Now, the first unscheduled task in the task sequence is selected and scheduled on a processor that can finish its execution at the earliest using duplication. This algorithm uses insertion based scheduling policy which considers the possible insertion of a task/ duplicated task in an earliest idle time slot (scheduling hole) between two already scheduled tasks on the processor. A task on the processor can start execution only after the data arrived from all of its immediate predecessors. The parent of task n_i whose data arrives last of all is termed as the most important immediate parent (MIIP). Data arrival time for task n_i on processor p_k is given by:

$$DAT(n_i, p_k) = \max_{n_j \in pred(n_i)} \{ \min \{ F_{jk}, F_{jk} + \varepsilon_{jk} \} \} \quad (3.18)$$

where F_{jk} is the finish time of task n_j on processor p_k . ε_{jk} is the actual communication time of data transfer from task n_j to n_i . Here, p_k refers to a processor that hosts originally scheduled task n_j .

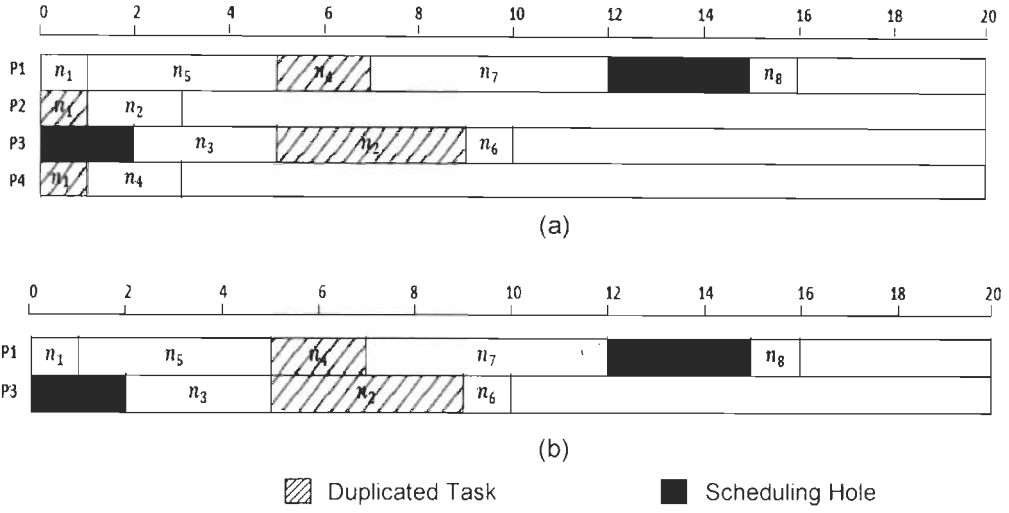


Figure 3.4: Gantt charts showing the schedules of (a) HLD (Duplications = 4, PC= 84.4%, P=4); (b) HED (Duplications = 2, PC=80.8%, P=2) for DAG shown in Figure 3.3

Due to the non-availability of data earlier, owing to precedence constraints or communication delay, a processor may remain idle leading to the formation of scheduling holes. These scheduling holes may be exploited to duplicate tasks to minimize data arrival time. The start time (S_{ik}) of task n_i on processor p_k is limited by the data arrival from its MIIP (say task M_i) and availability of a suitable scheduling hole. If suitable scheduling hole is not available then task n_i can start after the completion of last scheduled task on processor p_k , i.e., the ready time (p_k^R) of processor p_k . The start time of task n_i on processor p_k is given by:

$$S_{ik} = \max\{DAT(M_i, p_k), \min\{p_k^R, G_r^S\}\} \quad (3.19)$$

where G_r^S is the start time of first suitable scheduling hole G_r to accommodate task n_i on the processor p_k , if exist. The finish time (F_{ik}) is calculated as:

$$F_{ik} = S_{ik} + \omega_{ik} \quad (3.20)$$

The finish time is calculated for all the available processors and task n_i is scheduled on the processor that gives earliest finish time. After scheduling all the tasks, makespan is calculated as:

$$makespan = \max\{F_{ik}\} \quad \forall 1 \leq i \leq n \text{ and } 1 \leq k \leq p \quad (3.21)$$

Algorithm 3.2: HED

Begin

- 1: Construct a priority based task sequence ξ ;
- 2: *do* {
- 3: Select the first unscheduled task n_i in the task sequence ξ .
- 4: *for* (all p_k in processor list P) {
- 5: Compute finish time F_{ik} of n_i on p_k and sort the list of immediate parents of n_i in non-increasing order of data arrival time;
- 6: *for* all immediate parents, select the first immediate parent n_j from the list at Step 5 {
- 7: *if* duplication of n_j can reduce the finish time F_{ik} of task n_i on p_k
- 8: Duplicate n_j .
- 9: }
- 10: Compute the earliest finish time F_{ik} of n_i on p_k .
- 11: }
- 12: Find the minimum earliest finish time of task n_i .
- 13: Assign task n_i on processor p_k with minimum F_{ik} in schedule S ;
- 14: } *while* (there are unscheduled tasks in the task sequence ξ):
- 15: Maintain a list X of tasks which have been duplicated later and list Y of duplicated tasks in non-increasing order of their earliest start time;
- 16: *for* (each duplicated task n_i in list Y) {
- 17: *if* (no change in makespan of schedule S after removing duplicated task n_i)
- 18: Remove this duplicated task n_i from the schedule S and update list X ;
- 19: }
- 20: *for* (each task n_i in list X) {
- 21: *if* task n_i is unproductive task in schedule S due to its duplication; remove this task n_i from the schedule S
- 22: }

End

Figure 3.5: HED Algorithm

Further, we maintain a list X of tasks (original version of tasks in the schedule) and list Y of duplicated tasks (duplicated version of tasks in the schedule) in non-increasing order of earliest start time. The above schedule is modified only if the removal of the duplicated task from list Y does not affect the makespan adversely. Similarly, the tasks in list X that are unproductive are removed from the schedule. This modified schedule contains lesser number of duplications and remarkably less processor consumption as compared with HLD and LDBS for heterogeneous computing systems.

Gantt charts for the schedule generated by HLD and HED have been shown in Figure 3.4. In the schedule generated by HLD, task n_2 on processor p_2 and task n_4 on processor p_4 are unproductive after being duplicated on processor p_3 and p_1 respectively. Hence in HED schedule, tasks n_1 (which was scheduled on p_2 for n_2), n_2 from processor p_2 and task n_1 (which was scheduled on p_4 for n_4), n_4 from processor p_4 have been removed. It shows that HED uses less duplications and lesser number of processors with reduced processor consumption as compared to HLD for the same makespan.

The RD considers the same scheduling policy for homogeneous system of multiprocessors as described in HED (Figure 3.5). In this, a task sequence is generated using critical path based priority (Section 3.2.5(a)). A task sequence $\{n_1, n_7, n_9\}$ constructs the CP for the DAG shown in Figure 3.6. To satisfy the precedence constraints, predecessor nodes are added in the CP sequence with priority decided on higher b-level and ties being broken on the basis of lower t-level to get a critical path based task sequence $\{n_1, n_3, n_2, n_7, n_6, n_5, n_4, n_8, n_9\}$ for the DAG (Figure 3.6). Since the execution costs of all processors and the data transfer rates (bandwidths) between the processors are same in the homogeneous computing environment, the tasks are scheduled on to the processors that give minimum earliest start time instead of earliest finish time using an insertion-based duplication scheme. The pseudo code for the RD algorithm is same as HED except that a task sequence is generated using

critical path priority instead of using b-level and tasks are assigned on to the processor that minimizes earliest start time (S_{ik}) instead of earliest finish time (F_{ik}).

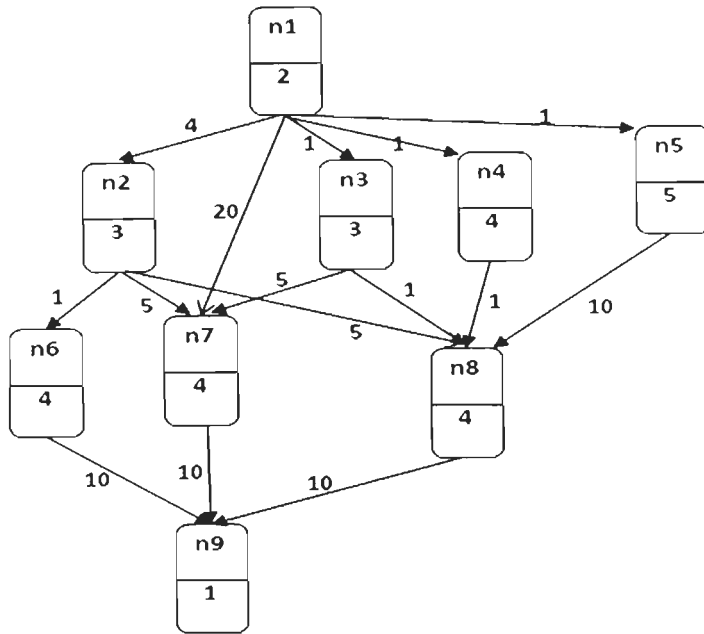


Figure 3.6: A simple DAG with precedence constraints

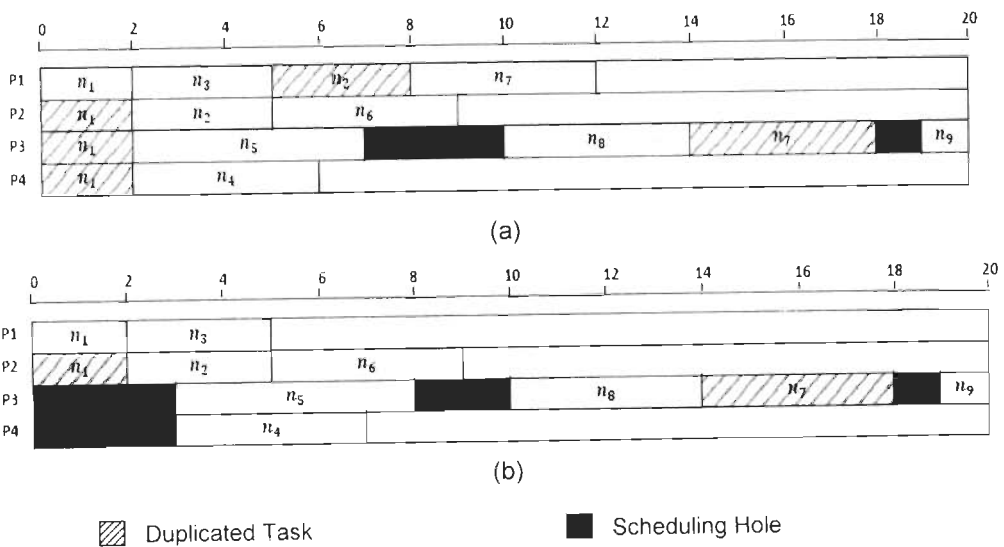
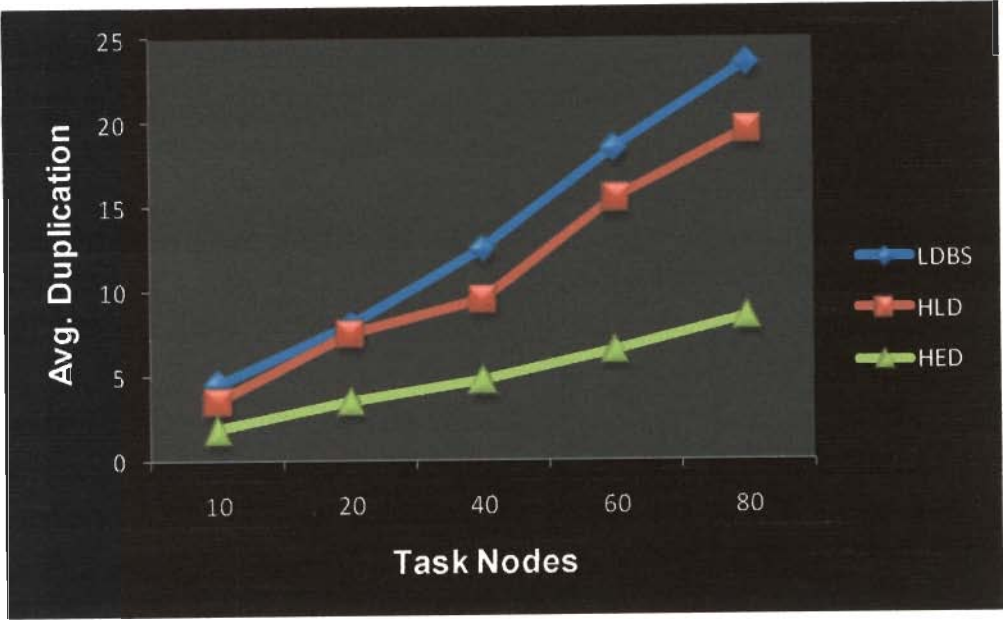


Figure 3.7: Gantt charts showing the schedules of (a) SD (Duplications = 5, PC = 91.5%); (b) RD (Duplications = 2, PC = 78%) for DAG shown in Figure 3.6

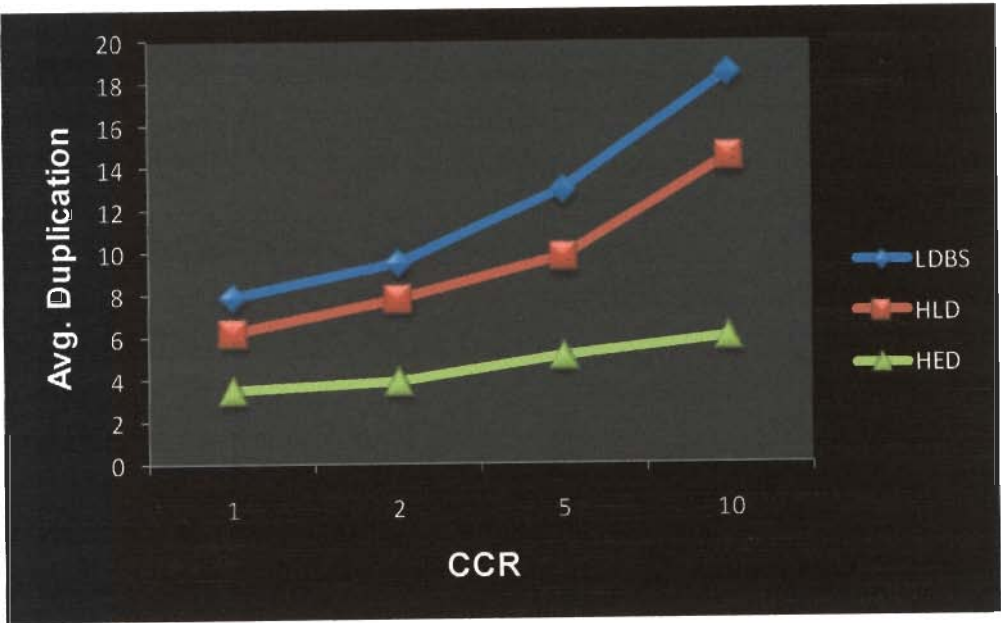
An application DAG is illustrated in Figure 3.6 for homogeneous computing system. Gantt charts for the schedule generated by SD and RD algorithms for this application DAG are presented in Figure 3.7. In SD schedule, task n_7 on processor p_1 is unproductive after being duplicated on processor p_3 , while in RD schedule, both tasks n_2 (which was scheduled on p_1 for n_7) and n_7 have been removed from processor p_1 . The SD schedule has been further modified without affecting the makespan after removing duplicated copies of task n_1 on processors p_3 and p_4 . It indicates that RD algorithm generates a better schedule as compared to SD algorithm with lesser number of duplications and remarkably lesser processor consumption.

3.3.4 Performance Comparisons and Result Analysis

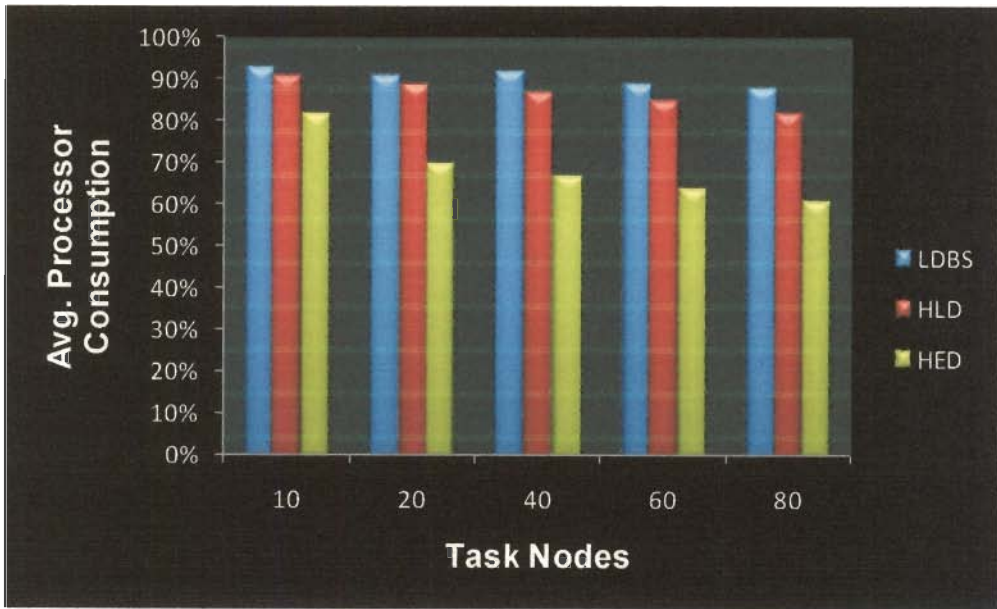
Simulation results for the random task graphs are presented in Figures 3.8 and 3.9 for the clique topology for different task graph sizes and CCRs. The performances of the proposed scheduling algorithms (HED and RD) have been evaluated with respect to various workflow application characteristics (task sizes, CCRs). The experimental results have been compared in terms of average number of duplications and processor consumptions for heterogeneous computing systems (Figure 3.8) and homogeneous computing systems (Figure 3.9). Each result obtained, with respect to CCR, is an average of 25 graphs (over 5 sizes and 5 average parallelisms), and with respect to graph size, is an average of 20 graphs (over 4 CCRs and five average parallelisms). In these experiments, the HED outperforms the HLD, LDBS in heterogeneous computing environments, and RD outperforms the SD, CPFD in homogeneous computing environments for different workflow sizes and CCRs.



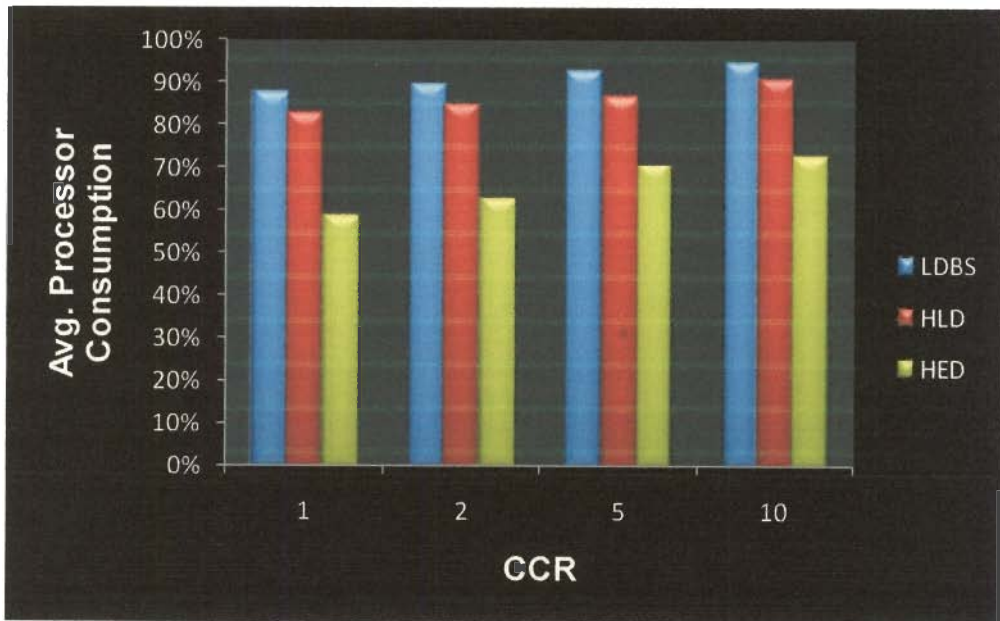
(a)



(b)

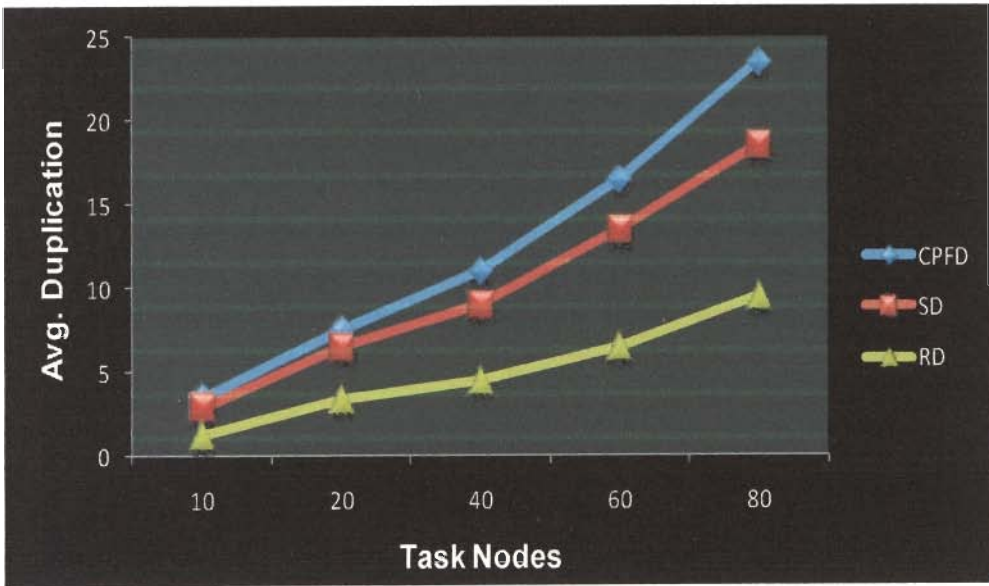


(c)

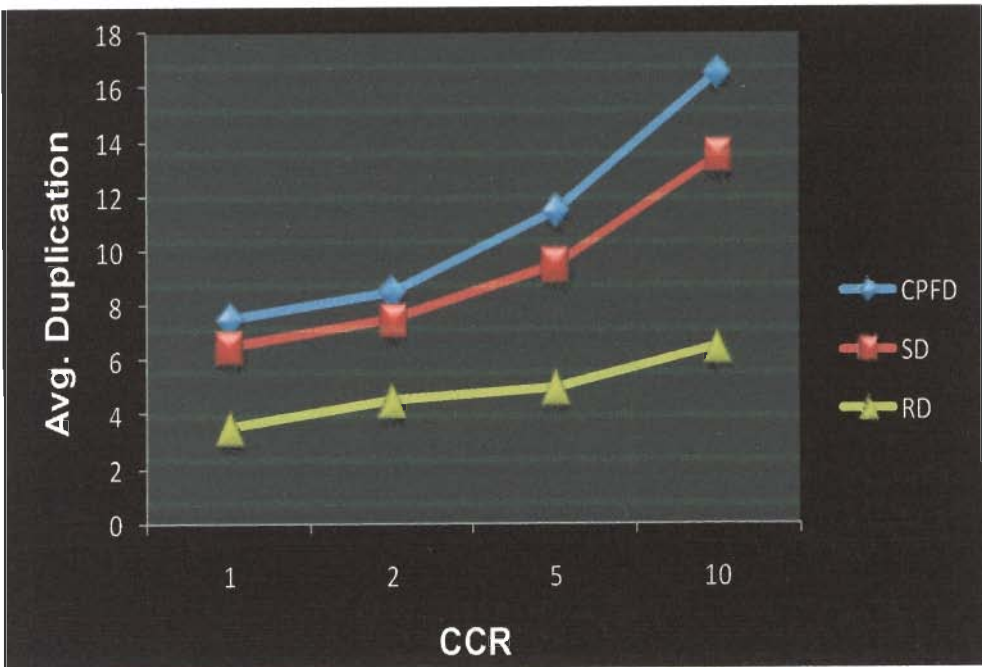


(d)

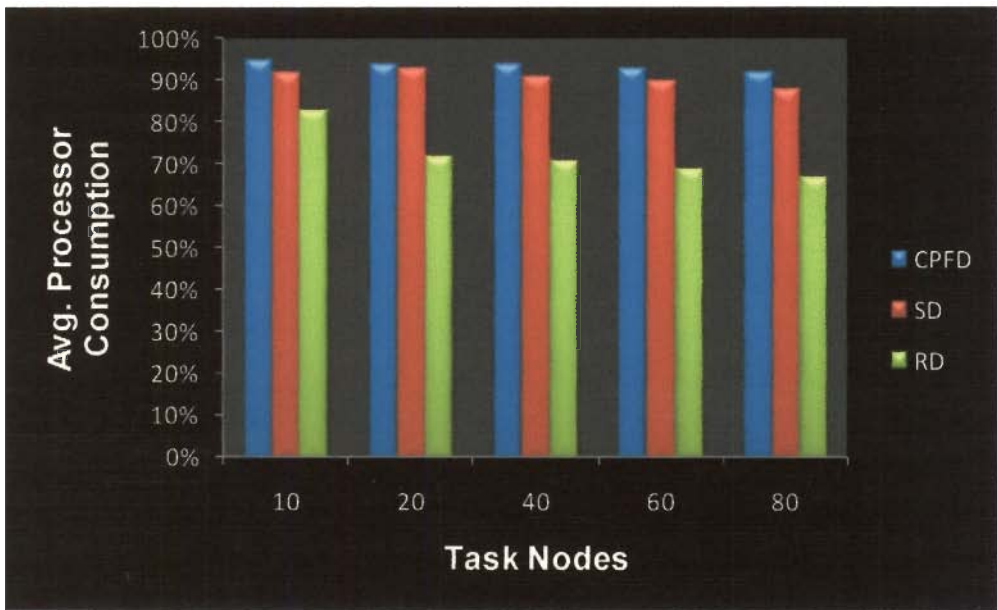
Figure 3.8: (a) to (d): Performance comparison of HED on random graph suite for heterogeneous systems



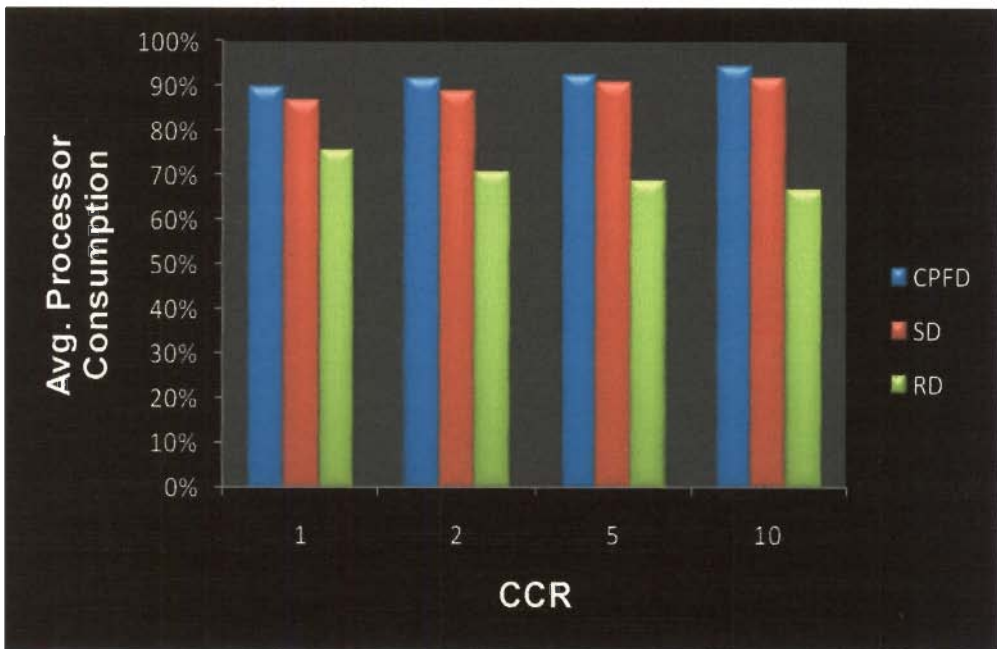
(a)



(b)



(c)



(d)

Figure 3.9 (a) to (d): Performance comparison of RD on random graph suite for homogeneous systems

3.4 Summary and Discussions

Heterogeneity of computing and network resources plays an important role in scheduling of workflow tasks over the heterogeneous computing environments. Tasks are scheduled in order of task sequence generated using b-level. In SHCP, we have formulated computation and network heterogeneity factors to obtain the task sequence. SHCP algorithm deviates the task sequence obtained using b-level. It efficiently considers the heterogeneity in resources and schedules the tasks on to the resources which produce better makespan. The experimental results show that SHCP generates shorter schedules for large workflow applications.

A duplication based strategy has been found momentous for homogeneous and heterogeneous computing systems to improve the performance. Duplication improves performance and reliability of such systems by duplicating critical tasks with higher communication costs. In HED and RD algorithms, the impact of duplications over makespan has been analyzed and the schedules are further improved by eliminating some duplicated and unproductive tasks as much as possible without affecting the makespan. These scheduling algorithms show that they are very useful in the distributed systems to reduce the scheduling cost and improving the performance of workflows.

The approaches presented optimize schedule by reducing duplications as much as possible without affecting the makespan and improve the system performance so that application execution cost and duplication overhead can be reduced. Performance comparisons with best known duplications algorithms such as CPF and SD for homogeneous system and LDBS and HLD for heterogeneous computing system show that RD and HED scheduling algorithms generate comparable schedules with remarkably less duplications and less processor consumption.

4.1 Overview

Multi-objective formulations are realistic models for many complex engineering optimization problems [14, 54]. The objectives are generally conflicting, preventing simultaneous improvement of each objective. Most real world engineering problems do have multiple objectives, i.e., minimize cost, maximize performance, maximize reliability, etc. In general, we can separate the optimization criteria of scheduling into two categories: those relating to completion time and those relating to monetary cost. The matter of time is concerned with the total execution time of whole workflow whereas the cost is about the monetary cost of resource usage and cost associated with waiting time of operations before and/or after they are processed.

“*How to combine these objectives?*” is a critical matter. One approach is to combine all objectives into a single objective function and then try to optimize the combined objective function. This leads to so called *Pareto optimality* [130], where an entire solution space is spanned. The other approach is to fix one or more parameters before hand and optimizes the other parameter. The second approach can be useful because the different

parameters are generally conflicting and incomparable. Thus, in bi-criteria scheduling, one parameter is optimized and then allowing a slack in its value, the other parameter is optimized to get a final schedule [84, 130]. In this research work, the second approach has been explored for developing a scheduling algorithm for workflows where duplication is adopted in a selective manner to optimize one objective.

In general, minimization of total execution time or makespan of the schedule is considered as the most important scheduling criteria [9, 37, 106, 124]. The convergence of Grid computing towards a service-oriented approach (Utility Grid) is fostering a new vision where economic aspects have become other equally important criteria [100, 108]. Utility Grids enable users to consume utility services transparently over a secure, shared, scalable and standard world-wide network environment and pay for access services based on their usage and the level of QoS provided. In such 'pay-per-use' Grids, workflow execution cost (economic cost) must be considered during scheduling. In several economic market models [42, 78, 66], economic cost has been considered as an important scheduling criterion.

Therefore, our research interest has been directed towards multi-objective optimization approach in scheduling the application tasks. Since, these objectives are conflicting, therefore it is an issue to design an efficient multi-objective scheduling algorithm. In this chapter, we have proposed two bi-criteria scheduling algorithms (SODA and DBSA), for executing workflow applications in grids.

4.2 Compaction Based Bi-Criteria Scheduling

4.2.1 Preamble

In general, minimization of makespan of an application schedule is the most important scheduling criteria. Most of the existing grid computing systems are based on system-centric policies whose objectives are to optimize the system-wide metrics of performance i.e., makespan. The other factor to be considered simultaneously is the economic cost to take care of user-centric policies [42, 100], since different resources, belonging to different organizations, may have different policies for payments.

A majority of problems addressed in the literature show that the schedulers have been generated keeping a single criterion [9, 37, 106, 124]. Considering multiple criteria enables us to propose a more realistic solution. Therefore, an efficient multi criteria scheduling heuristic is required for execution of workflow on Grid while assuring the high speed of communication, reducing the tasks execution time and economic cost. As the DAG scheduling problem in Grid is NP-hard, we have emphasized on heuristics for scheduling rather than the exact methods. In literature, many bi-criteria scheduling algorithms have been proposed [18, 36, 78, 125, 130, 138], which minimize both makespan and economic cost of the schedule but only few of them address the workflow type of applications.

Tsiakkouri et al. [125] proposed a scheduling algorithm for DAGs in heterogeneous environments under budget constraints. Yu and Buyya [137, 138] proposed the multi-objective planning for workflow scheduling approaches for utility Grids. Dogan and Ozguner [36] gave another trade-off between makespan and reliability using a sophisticated reliability model assuming computation and network performance. In [78], quality of service (QoS) optimization strategy for multi criteria scheduling on the Grid has been presented for QoS criteria namely payment, deadline and reliability, but they

do not address the workflow type of applications. We propose an approach where processor requirement is minimized under two criteria (i.e., makespan and economic cost) for workflows. The study also shows that heuristics performing best in static environment [9, 106] have the highest potential to perform better in more accurately modeled Grid environment.

Scheduling heuristics proposed in the literature offer trade-offs between the quality and the time complexity of schedules. Most existing multi criteria scheduling approaches adopted the list scheduling heuristics [113, 124] as a primary scheduling, which is a low time complexity technique that produces relatively low quality solutions. In our survey, it has been observed that duplication based heuristics [9, 37] generate remarkably much shorter schedules as compared with the list and clustering heuristics by assigning some of the tasks redundantly on multiple resources reducing inter-processor communication.

The duplication strategy enables us to utilize the idle time-slots which are rarely occupied by other application tasks. Comparative schedules of an application DAG are presented in Figure 4.1 to illustrate our assertion about duplication-based heuristic [9, 18] over list scheduling heuristic [113, 124]. In Figure 4.1 (b), computation cost matrix represents the execution time of tasks on processors. The computation cost 'infinity' indicates that some tasks cannot be executed on some processors. This motivates us to adopt an efficient duplication based approach for primary scheduling to generate the initial schedule which gives 'makespan' (primary criterion) much lesser than any other list scheduling heuristic.

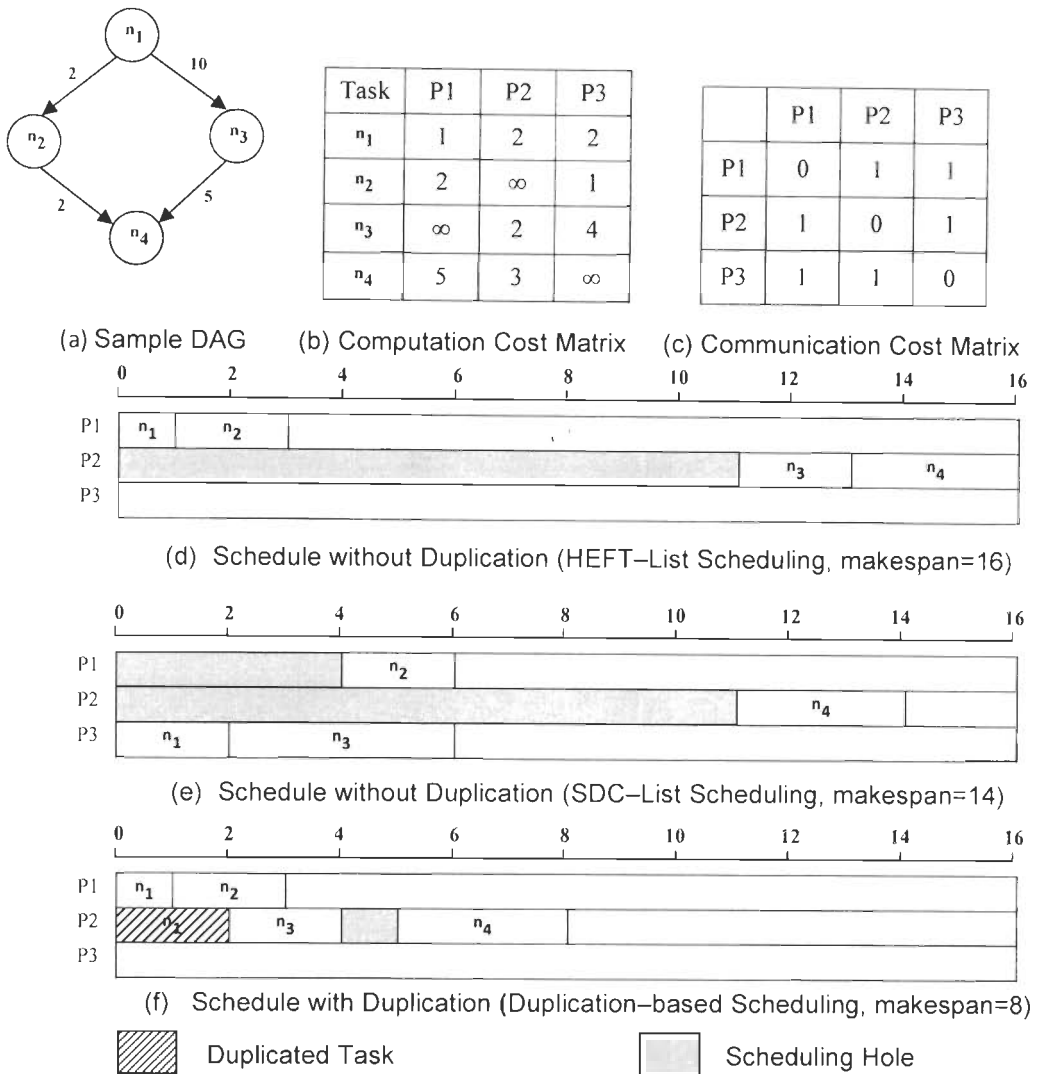


Figure 4.1: Effectiveness of duplication based scheduling over list scheduling

In literature [125], two approaches i.e., LOSS and GAIN were proposed to compute the weight values for a given DAG. In LOSS, initial assignment is done for optimal makespan using an efficient DAG scheduling [124] whereas, in GAIN, initial assignment is done by allocating tasks to cheapest machines in order to reduce the economic cost as much as possible. We consider the LOSS approach where initial assignment is done using duplication-based scheduling approach inspired from our earlier work discussed in Section 3.3 rather than HEFT [124] since duplication-based

heuristic produces shorter makespan. In secondary scheduling, we target the processor requirements by removing redundant duplicated tasks from schedule and thereby optimize the 'economic cost' (secondary criterion) of the schedule without affecting the makespan obtained in primary scheduling. In this work, LOSS approach [125] has been used to reduce the economic cost in secondary scheduling.

4.2.2 Grid Resource Model

A Grid resource model has already been discussed in Section 3.2.2. The processing capacities of resources (in MIPS) and machine price per MIPS execution are shown by Tables 4.1 and 4.2 for the sample grid resource model illustrated in Figure 4.2. In Figure 4.2, the solid lines indicate the direct path between resources while the dotted lines show that there is no direct path and communication between these resources is done through alternative paths. The values on these lines are showing the maximum data transfer rates (or bandwidth) available between the grid resources. Table 4.1 presents the resource capacity (computation speed of processor in terms of MIPS rating) of executing tasks (in MIPS rating). Table 4.2 shows the monetary execution cost (in grid Dollar) of grid resource per MIPS for executing tasks.

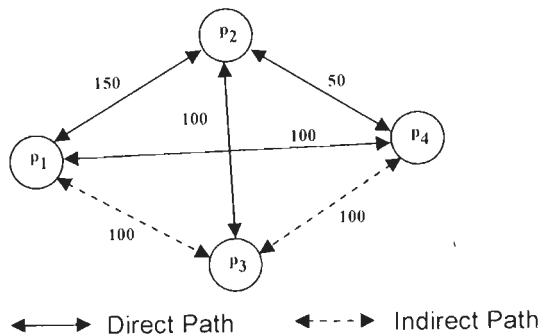


Figure 4.2: A sample Grid consists of four resources (Bandwidths are in Kbps)

Table 4.1: Resource capacity

Resources	P1	P2	P3	P4
Processing Capacity, $\alpha(p_i)$ (in MIPS)	220	350	450	310

Table 4.2: Machine price

Resource p_i	Machine cost per MIPS M_i (in g\$)
1	1.0
2	2.5
3	3.0
4	2.0

4.2.3 Workflow Application Model

In the context of Grid computing, there exist several applications such as bioinformatics, financial analysis etc. that can be constructed as workflows. A scheduling problem can be defined as the assignment of different Grid services to different workflow tasks. Every workflow can be modeled as DAG, as shown in Figure 1.1. The application model has already been discussed in Section 3.2.3.

Most of the existing scheduling algorithms consider that all the available resources are equally capable, i.e., each resource can execute all the tasks with possibly different processing rates. While some of them do not make the assumption explicitly as they do not consider the potential effect of different capabilities [106, 124]. Thus, these algorithms suffer in performance and may become inapplicable without modification such as CPOP [124] assign all tasks onto a single processor in an attempt to minimize the execution time of critical tasks. This algorithm fails if none of the processor is able to execute all the tasks (see Table 4.3). The computation cost of task n_i

on resource p_j is ω_{ij} . If the resource p_j is not capable to process the task n_i , then $\omega_{ij} = \infty$.

Table 4.3: Computation cost, b-level and task sequence for DAG in Fig. 1.1

Task Node	Computation Cost ω_{ij} (in msec)				Mean Time (ms)	b-level	Task Sequence
	p_1	p_2	p_3	p_4			
n_1	1	1	∞	1	1	35	n_1
n_2	∞	2	4	3	3	19	n_5
n_3	5	∞	3	4	4	20	n_4
n_4	2	∞	5	2	3	28	n_3
n_5	4	8	∞	6	6	29	n_2
n_6	3	∞	1	2	2	9	n_7
n_7	5	5	∞	5	5	15	n_6
n_8	1	3	2	∞	2	2	n_8

The optimization goal of bi-criteria scheduling is to obtain the schedule with minimum schedule cost. It can be expressed in terms of performance metrics called effective schedule cost (*ESC*) which can be computed using Equation (4.6). Another performance metrics is normalized schedule length (*NSL*) (see Equation (4.4)) of the schedules by our approach and compared with the existing bi-criteria scheduling algorithms.

4.2.4 Performance Metrics

The proposed bi-criteria algorithm focuses on two objectives: makespan and economic cost. The makespan is the total time between finish time of exit task and the start time of entry task in the given DAG. The economic cost (*EC*) is the summation of the economic costs of all workflow tasks scheduled on different resources which is computed as:

$$EC = \sum_{j=1}^m D_j \quad (4.1)$$

where m is the total number of available resources in the Grid and D_j is the execution cost of the tasks scheduled on a resource p_j , D_j is given as:

$$D_j = PBT_j \times \alpha(p_j) \times M_j \quad (4.2)$$

where M_j is the per MIPS machine (processor) cost (in grid Dollar or g\$) of executing task on a resource p_j , and PBT_j is the total busy time consumed by tasks scheduled on a resource p_j . In this model, the cost of idle time slots between the scheduled tasks on any resource is also considered in the economic cost as it is difficult for the grid scheduler to schedule other workflow tasks in these idle time slots. Thus, the total execution time (makespan) can be expressed as:

$$makespan = AFT(n_{exit}) - AST(n_{entry}) \quad (4.3)$$

where AFT and AST are the actual finish and actual start time of the exit task and the entry task respectively. The normalized schedule length (NSL) of a schedule can be calculated as:

$$NSL = \frac{makespan}{\sum_{n_i \in CP_{min}} \min_{p_j \in P} \{\omega_{ij}\}} \quad (4.4)$$

The denominator is the summation of the minimum execution costs of tasks on the CP_{min} [124] in the given DAG. In the Grid, some resources may not always be in a fully connected topology. Therefore, bandwidths between such resources can be computed by searching alternative paths between them with maximum allowable bandwidths. The communication cost between

the task n_i scheduled on the resource p_m and the task n_j scheduled on the resource p_n can be computed as:

$$\varepsilon_{ij} = \frac{c_{ij}}{\beta(p_m, p_n)} \quad (4.5)$$

In this model, we avoid the communication startup costs of resources and intra-processor communication cost is negligible. A workflow of tasks is submitted to the Grid Scheduler [84] where tasks are queued in non-decreasing order of their b-level. The b-level (bottom level) of task n_i can be defined as the longest directed path including execution time and communication time from task n_i to the exit task in the given DAG. The optimization goal of bi-criteria scheduling is to obtain the schedule with minimum schedule cost. It can be expressed in terms of performance metrics called effective schedule cost (*ESC*) which can be computed as:

$$ESC = NSL \times EC \quad (4.6)$$

4.2.5 SODA Algorithm

In this work, a fairly static methodology has been adopted for defining the weights of the computational tasks and communicating edges. The 'execution time (makespan)' is the total time between the finish time of exit task and start time of the entry task in the given DAG. Similarly, the 'economic cost' (EC) is the summation of the economic costs of all workflow tasks scheduled on different resources which can be calculated using Equation (4.1).

In our model, the cost of the idle slots between the scheduled tasks on any resource is also considered as part of economic cost as it is difficult for the Grid scheduler to schedule other workflow tasks in these idle slots. After

assignment of all tasks in a DAG on Grid resources, the makespan of the schedule will be the actual finish time of the exit task n_{exit} which can be computed using Equation (4.3). Since a large set of task graphs with different properties is used, it becomes necessary to normalize the schedule length (makespan) to a lower bound, called the normalized schedule length (*NSL*) which is calculated using Equation (4.4). A resource management and scheduling model is shown in Figure 4.3.

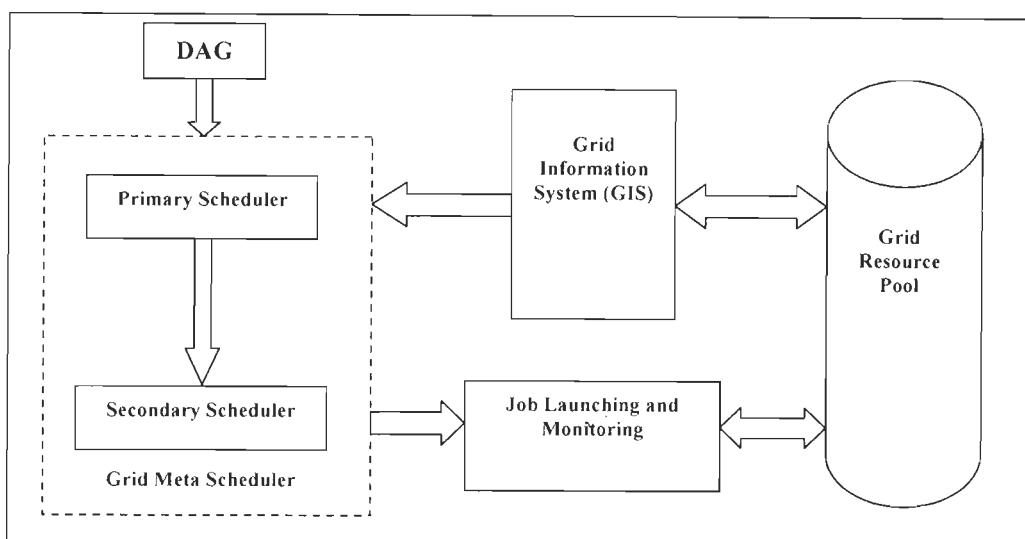


Figure 4.3: A Grid resource management and scheduling model (GRMS)

In this algorithm, we consider the makespan as the primary criterion and economic cost as the secondary criterion. The primary scheduler (Figure 4.3) is used to generate a schedule that minimizes the makespan. The scheduler adopts an efficient duplication scheduling approach to optimize the makespan as much as possible. Further, this schedule is forwarded to the secondary scheduler (Figure 4.3) while the primary scheduler simultaneously gets another workflow for scheduling. Therefore, the primary and secondary scheduler can work in parallel on different workflows. The secondary scheduler optimizes the primary schedule to minimize the number of processors and thus economic cost subsequently.

Algorithm 4.1: Primary Scheduling in SODA

(1) **Input:** A workflow (w) or DAG with task computation and communication costs. Set of available resources P along with the cost of execution per unit time.

(2) Construct a priority based task sequence ξ based on highest b-level first.

(3) **For** (each unscheduled task n_i in the task sequence ξ)

Assume the finish time F_i of task n_i is Infinite.

For (each capable resource p_j)

(3a) Compute finish time F_i of task n_i on resource p_j .

(3b) Construct task predecessor list $\text{pred_list}(n_i)$.

(3c) Initialize $\text{temp_list}(n_i, p_j)$ to zero.

(3d) **If** (pred_list)

For (each predecessor d_k not scheduled on p_j)

If duplication of d_k on p_j reduces the finish time F_{ij}

Add d_k in $\text{temp_list}(n_i, p_j)$.

Update finish time F_{ij} .

EndIf

EndFor

EndIf

(3e) **If** $F_{ij} < F_i$

$F_i = F_{ij}$.

$r_i = p_j$.

If (temp_list)

Copy $\text{temp_list}(n_i, p_j)$ into $\text{duplicate_list}(n_i, p_j)$.

EndIf

EndIf

EndFor

Assign task n_i on resource r_i and update the schedule S .

If (duplicate_list)

Duplicate tasks from duplicate_list to r_i and update schedule S .

EndIf

EndFor

(4) Compute makespan c_1^{prel} (Eq. 4.3) and economic cost c_2^{prel} (Eq. 4.1) from schedule S .

Figure 4.4: The pseudo code of primary scheduling in SODA

Algorithm 4.2: Secondary Scheduling in SODA

(1) **Input:** A workflow (w) or DAG with task computation and communication cost, set of available resources P along with the cost of execution per unit time. Consider makespan c_1^{prel} and economic cost c_2^{prel} obtained from primary scheduling using Algorithm 4.1 (Figure 4.4). A schedule S and duplicate_list from Algorithm 4.1.

(2) **If** (duplicate_list)

(2a) Copy tasks from duplicate_list into list A and sort them in non-decreasing order of their start time.

(2b) **For** (each duplicated task a_i in A)

Compute schedule length SL without considering a_i in schedule S .

If ($SL \leq c_1^{prel}$)

$c_1^{final} = SL$.

Remove a_i from schedule S and update list A .

EndIf

EndFor

(2c) Construct list B of tasks from list A that were duplicated on other resources.

(2d) Sort list B in non-decreasing order of task start time.

(2e) **For** (each task b_i in B)

Compute schedule length SL without considering b_i in schedule S .

If ($SL \leq c_1^{prel}$)

$c_1^{final} = SL$.

Remove b_i from schedule S and update list B .

EndIf

EndFor

EndIf

(4) Compute economic cost c_2^{final} of the optimized schedule S .

(5) **For** (each task n_i scheduled on resource p_j in schedule S)

(5a) Construct list R of capable resources in non-decreasing order whose machine cost is less than M_j (Table 4.2).

(5b) **For** (each alternative resource p_k in R)

i. Reschedule task n_i to resource p_k for $c_1^{final} \leq c_1^{prel}$.

ii. Compute economic cost EC' (Eq. 4.1).

iii. **If** ($EC' < c_2^{final}$)

Update schedule S .

$c_2^{final} = EC'$.

EndIf

EndFor

EndFor

Figure 4.5: The pseudo code of secondary scheduling in SODA

We have adopted three phases to balance the makespan and economic cost in secondary scheduling:

1. Removal of useless duplicated tasks
2. Removal of unproductive schedules
3. Shifting of tasks to cheaper resources without affecting makespan

First, it investigates the duplicated tasks in the schedule and modifies the schedule after removing those duplications whose removal keeps the makespan unaffected. Second, it analyses those tasks in the schedule which have been duplicated on other resources. Sometimes, such tasks or schedules may become unproductive if their descendent tasks are receiving input data from their duplicated version. Thus, such unproductive tasks or schedules are removed in order to minimize the processor requirements and reduce the economic cost. Then, each task in the above schedule is tried to reschedule from current processor to other capable processor in order to minimize the economic cost while keeping the makespan of the schedule unaffected.

The pseudo code of the proposed algorithm (SODA) is described in Algorithm 4.1 (Figure 4.4) and Algorithm 4.2 (Figure 4.5). It is divided into two major phases:

1. Primary scheduling – optimization for the primary criterion only
2. Secondary scheduling – minimization of processor requirements and optimization for the secondary criterion without affecting the makespan obtained in primary scheduling.

An efficient duplication-based heuristic has been applied for the primary scheduling for optimizing the primary criterion (makespan). It

generates a preliminary solution $Sol_w^{Prel} \in SC$ with the total costs of primary criterion (i.e., makespan) and the secondary criterion (i.e., economic cost) which can be denoted as c_1^{Prel} and c_2^{Prel} , respectively. The set SC contains all possible schedules for workflows to be executed [130].

4.2.6 Performance Comparisons and Result Analysis

The SODA algorithm has been implemented in MATLAB using TORSCHE scheduling toolbox (see Appendix – A) for the evaluation of different random task graphs or DAGs of different graph sizes (100, 200, 300, 400, 500) and different parallelisms (2, 4, 6, 8, 10). The proposed approach has been compared with DCA [130]. The algorithms have been executed and compared in a Grid of different resource size (20, 40, 60, 80, 100). The SODA algorithm has been compared with DCA for performance evaluation of effective schedule cost (ESC), economic cost and NSL with respect to various random workflows of different sizes. The algorithms have been run under the same conditions for fair comparison: for each workflow, each algorithm is run to find best possible second criteria cost without altering the makespan in primary scheduling.

The simulated results and graphs reveal that the proposed bi-criteria scheduling approach (SODA) outperforms the DCA algorithm in terms of both economic cost and schedule length. In Figure 4.6, SODA algorithm yields reduced economic cost (EC) as compared to DCA for different workflow applications. Figure 4.7 depicts the improvement of SODA over DCA in terms of effective schedule cost for different workflows. Further, SODA algorithm performs better in terms of average NSL (Figure 4.8) for the schedules generated in secondary scheduling. As duplication strategy has been adopted in an efficient manner, our approach shows improvement over the DCA algorithm to be adopted for grid scheduling.

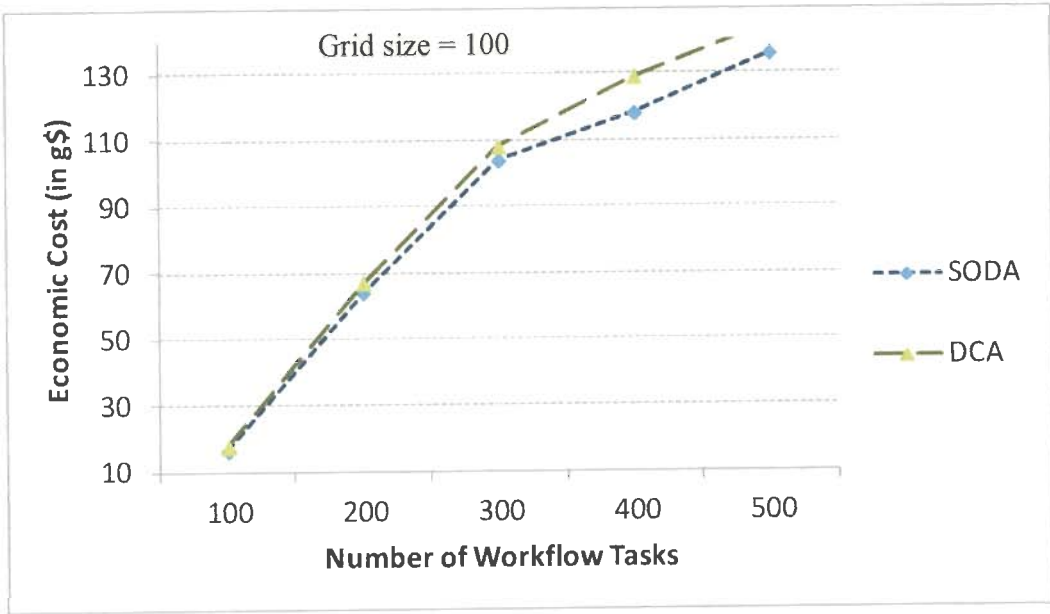


Figure 4.6: Effect of workflow size on economic cost

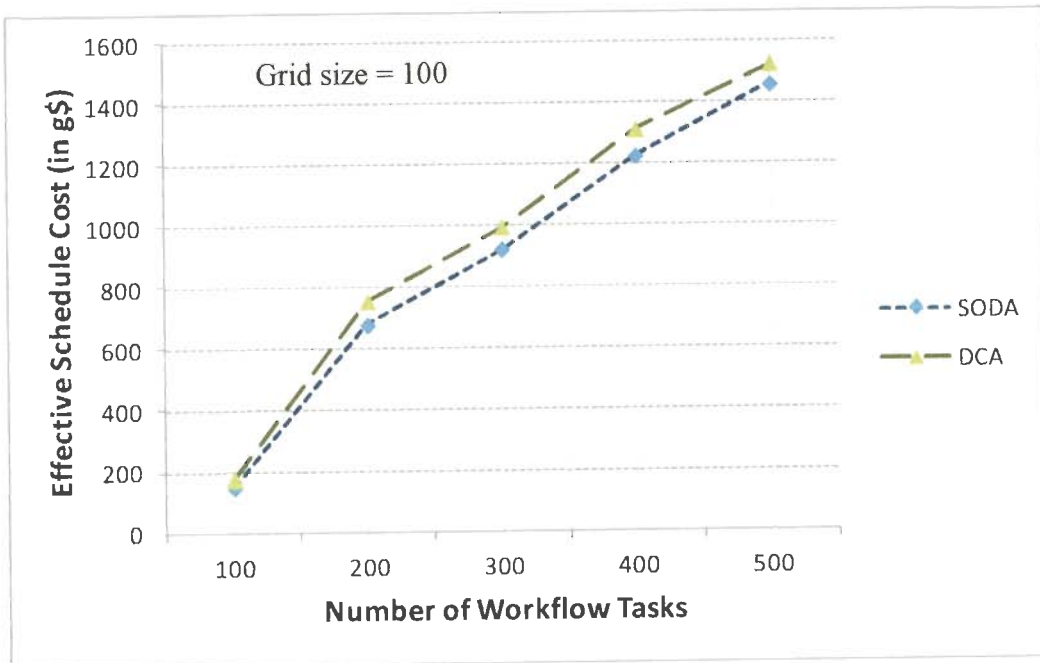


Figure 4.7: Effect of workflow size on effective schedule cost

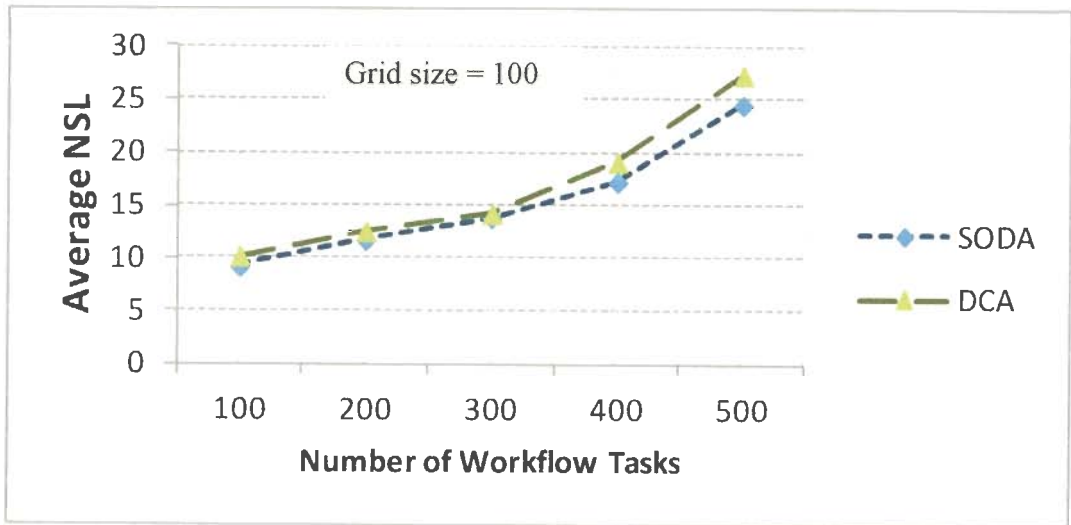


Figure 4.8: Effect of workflow size on Average NSL

Table 4.4: Grid simulation environment layouts

Number of grid resources	[20, 100]
Resource Bandwidth	[100 Mbps, 1 Gbps]
Number of tasks	[100, 500]
Computation cost of tasks	[5 msec, 200 msec]
Data Transfer Size	[20 Kbytes, 2 Mbytes]
Resource Capability (MIPS)	[220, 580]
Execution Cost (Per MIPS)	[1-5 grid dollar per MIPS]

In this research, a novel bi-criteria workflow scheduling approach has been presented and analyzed. We have proposed an efficient scheduling algorithm called 'schedule optimization with duplication-based bi-criteria scheduling algorithm' (SODA) which optimizes the makespan and economic cost of the schedule and minimizes the requirements of processors. The schedule generated by SODA algorithm is better than other related bi-criteria algorithms in respect of both makespan and economic cost. This algorithm decouples processor economization from schedule length minimization. In

next section, we extended this approach using sliding constraints for yielding much better schedules.

4.3 Two Phase Bi-Criteria Scheduling

4.3.1 Preamble

This section extends the previous bi-criteria research work by considering sliding constraints. It introduces a new bi-criteria scheduling heuristic called *Duplication-based Bi-criteria Scheduling Algorithm* (DBSA) for Grid computing environments. The proposed scheduling approach works in two phases: (1) Duplication-based Schedule Optimization for the primary criterion i.e., execution time, (2) Sliding Constrained Schedule Optimization – Optimizes secondary criterion i.e., economic cost while keeping primary criterion within the allowable slack (sliding constraint). The *sliding constraint* can be defined as a function of the primary criterion to determine how much the final solution can differ from the primary solution for primary criteria. The experimental results show that the proposed approach generates schedules which are fairly optimized for both economic cost and makespan while keeping the makespan within defined constraints for executing workflow applications in the grid environment.

In [36, 78, 138, 139, 130], several scheduling algorithms have been proposed which minimize the makespan and the economic cost of the schedule but only few of them address the workflow type of applications. The research work [138, 139] presented the multi-objective planning for workflow scheduling approaches for utility grids. Wieczorek et al. [130] presented bi-criterion scheduling algorithm known as *Dynamic Constraint Algorithm* (DCA) based on a sliding constraint. It adopted list-based scheduling heuristic for primary scheduling. Our work takes a different approach for the two specific

criteria (i.e., makespan and economic cost) while utilizing the effectiveness of duplications in minimizing makespan in Grids which has not been explored in any of the related work till date.

Deelman et al. [33] described the three different workflow scheduling strategies namely, full-plan ahead scheduling, in-time local scheduling, and in-time global scheduling. In in-time local (or global) scheduling, scheduling decision for an individual task is postponed as long as possible and performed before the task execution starts (fully dynamic approach). In full-plan ahead scheduling, the whole workflow is scheduled before its execution starts (fully static approach). We have adopted full-plan ahead scheduling as it does not incur the run time overheads and the associated scheduling complexity. The research study shows that heuristics performing best in the static environments (e.g., HLD [9], HBMCT [106]) have the highest potential to perform better in a more accurately modeled grid environment.

In extensive literature survey, it has been observed that duplication-based heuristics [9, 37] generate remarkably much shorter schedules as compare to the list based and cluster based heuristics. The duplication approach utilizes the idle time slots (scheduling holes) for task duplication which, in turn, reduces the communication time. This motivates us to adopt duplication-based approach for primary scheduling to optimize the *makespan* (Primary criterion).

In secondary scheduling, the objective is to optimize the *economic cost* (Secondary criterion) of the schedule while keeping the makespan within a defined sliding constraint. Figure 4.9 illustrates that the primary solution (M1, C1) can be obtained considering primary criterion i.e., makespan in the primary scheduling that yields makespan of length M1 while the economic cost is C1. In secondary scheduling, the above schedule is optimized for the economic cost allowing the makespan to increase from M1 to M2 (M2 is the maximum allowable schedule length) that yields the schedule with makespan

M2 and the reduced economic cost C2. The sliding constraint approach generates schedules which are better both in terms of the execution time and the economic cost [130].

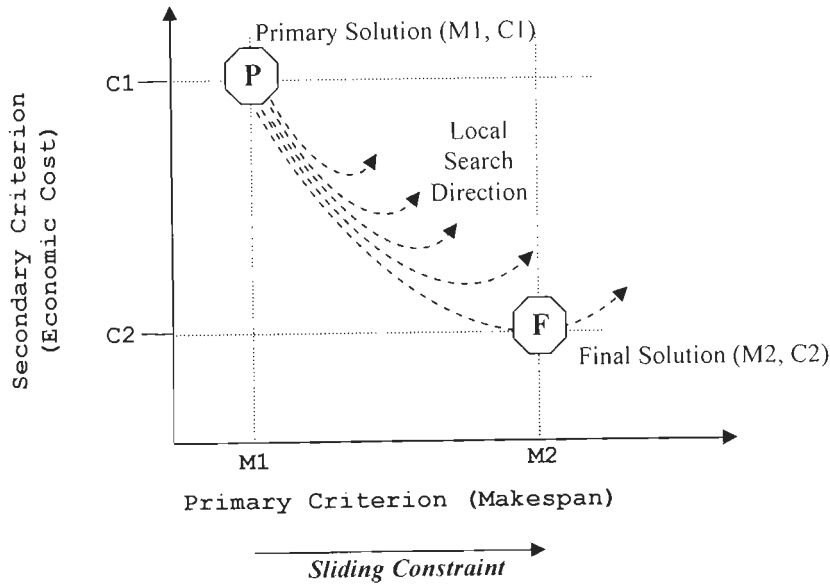


Figure 4.9: A bi-criteria optimization process

In general, bi-criteria optimization yields a set of solutions (a Pareto set) rather than a single solution. Each solution in a Pareto set is called Pareto optimum and when these solutions are plotted in the objective space they are collectively known as Pareto front. The main objective of multi criteria optimization problem is to obtain a Pareto front. In this research, we consider LOSS approach in secondary scheduling to optimize economic cost while increasing makespan within constraint limits.

4.3.2 Grid Resource Model

The Grid resource model is illustrated in Figure 4.2 which is already discussed in Section 3.2.2. The processing capacities of Grid resources have been shown in Table 4.1. In Table 4.2, machine execution prices per MIPS are given for the resources working in the Grid as illustrated by Figure 4.2.

4.3.3 Workflow Application Model

The workflow application model has been discussed earlier in Section 3.2.3. Table 4.3 shows the computation costs of workflow tasks on different processors and task sequence based on b-level computing for each task for application DAG shown in Figure 1.1. Some tasks may not be executed on some processors because processing time is infinite (e.g., n_1 on p_3 , n_2 on p_1).

4.3.4 DBSA Algorithm

In this scheduling approach, makespan is the primary criterion and economic cost is the secondary criterion. Sliding constraint for the primary criterion would represent how much the final solution may differ from the best solution found for the primary criterion in first phase of scheduling. The primary scheduler works on duplication-based scheduling approach to minimize the total schedule length (makespan) as much as possible. Then, this schedule is forwarded to the secondary scheduler. The secondary scheduler optimizes the above schedule to minimize the economic cost while letting makespan increase up to sliding constraint limit. In secondary scheduling, some duplicated tasks may be removed and the primary schedule is modified such that makespan of the schedule, after removing useless duplications, remains within maximum allowable execution length.

Algorithm 4.3: Secondary Scheduling in DBSA

(1) **Input:** A workflow (w) or DAG with task computation and communication cost, set of available resources P along with the cost of execution per unit time.
 Assume sliding constraint L (i.e. 10%, 25%, 50%, 75% of the makespan c_1^{prel} obtained from Algorithm 4.1).
 A schedule S and duplicate_list from Algorithm 4.1 (Figure 4.4).

(2) Let $SL_{New} = c_1^{prel} + 10\%$ of c_1^{prel} .

(3) **If** (duplicate_list && $c_1^{prel} \leq SL_{New}$)

(3a) Copy tasks from duplicate_list into list A and sort them in non-decreasing order of their start time.

(3b) **For** (each duplicated task a_i in A)

Compute schedule length SL without considering a_i in schedule S .

If ($SL \leq SL_{New}$)

$c_1^{final} = SL$.

Remove a_i from schedule S and update list A .

EndIf

EndFor

(3c) Construct list B of tasks from list A that were duplicated on other resources.

(3d) Sort list B in non-decreasing order of task start time.

(3e) **For** (each task b_i in B)

Compute schedule length SL without considering b_i in schedule S .

If ($SL \leq SL_{New}$)

$c_1^{final} = SL$.

Remove b_i from schedule S and update list B .

EndIf

EndFor

EndIf

(4) Compute economic cost c_2^{final} of the optimized schedule S .

(5) **For** (each task n_i scheduled on resource p_j in schedule S)

(5a) Construct list R of capable resources in non-decreasing order whose machine cost is less than M_j (Table 4.2).

(5b) **For** (each resource p_k in R)

i. Reschedule task n_i to resource p_k for $c_1^{final} \leq SL_{New}$.

ii. Compute economic cost EC' (Eq. 4.1).

iii. **If** ($EC' < c_2^{final}$)

Update schedule S .

$c_2^{final} = EC'$.

EndIf

EndFor

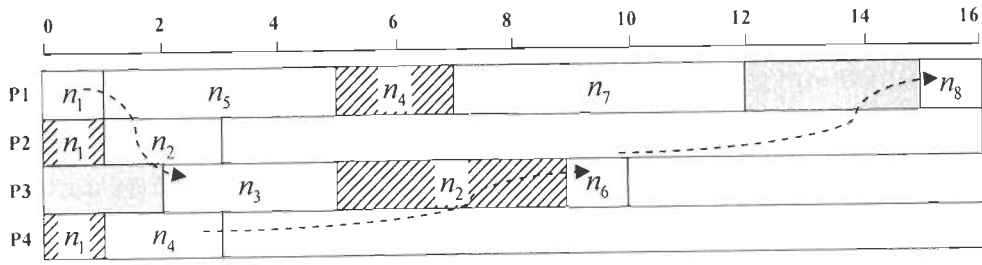
EndFor

Figure 4.10: The pseudo code for secondary scheduling in DBSA

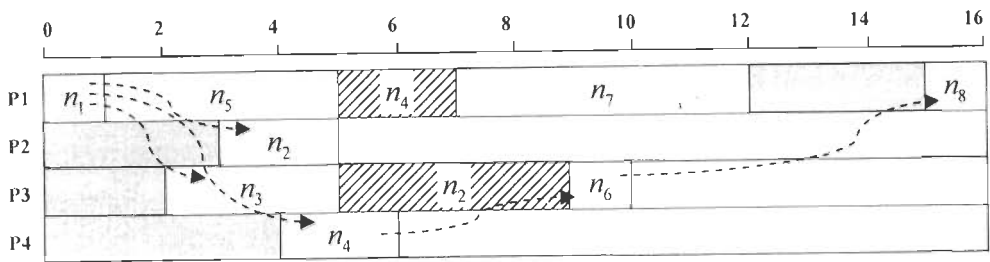
The proposed algorithm can be divided into two phases: (1) Primary Scheduling – optimizing the makespan, (2) Secondary Scheduling – optimizing economic cost, while keeping the makespan within the defined sliding constraint limit. An efficient duplication–based scheduling heuristic has been applied for the primary scheduling [102]. It generates a preliminary solution $sol_w^{prel} \in SC$, with the total costs of primary criterion and the secondary criterion denoted as c_1^{prel} and c_2^{prel} , respectively. The set SC contains all possible schedules for workflow (w) to be executed over the Grid [130].

The secondary scheduling optimizes the primary solution for the secondary criterion, generating the best possible solution $sol_w^{final} \in SC$ and the total costs c_1^{final} and c_2^{final} for primary and secondary criteria. The sliding constraint is equal to L such that the primary criterion cost can be increased from c_1^{prel} to $c_1^{prel} + L$. We can calculate the maximum allowable execution time T_{max} and minimum economic cost C_{min} of workflow application using cost optimization algorithm such as GreedyCost [138]. Similarly, maximum allowable economic cost C_{max} of a workflow with shortest possible execution time T_{min} can be computed using time optimization algorithm such as HED (see Section 3.3.3).

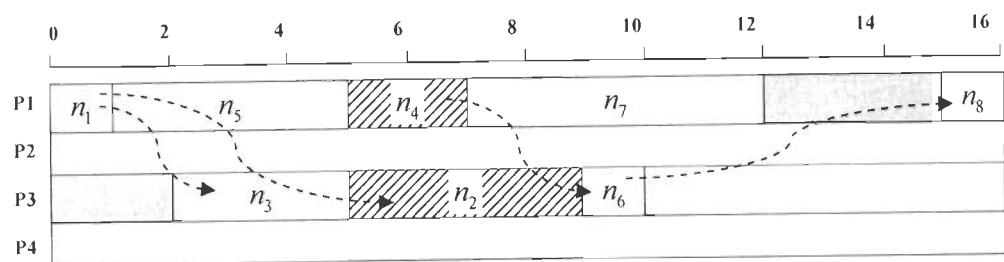
The schedule produced by primary scheduling is illustrated in Figure 4.11(a) for the workflow as shown in Figure 1.1. The makespan of this schedule is 16. This schedule yields the total economic cost of 18.81 g\$ computed as per machine cost described in Table 4.2 using Equations (4.1) and (4.2). Further, we apply the secondary scheduling to optimize the economic cost while keeping the makespan within maximum allowable limit considering sliding constraint.



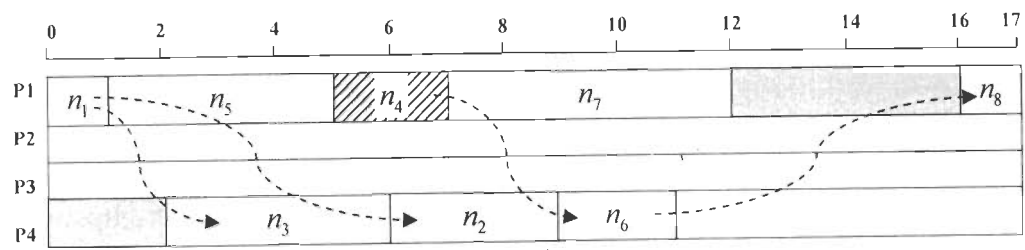
(a) Gantt chart showing a schedule by duplication-based scheduling (EC=18.81 g\$)



(b) Gantt chart showing a schedule after removing useless duplications (EC=17.31 g\$)



(c) Primary schedule after removing unproductive sub-schedules (EC=14.32 g\$)



(d) Gantt chart showing a schedule after applying Greedy approach (EC=9.32 g\$)

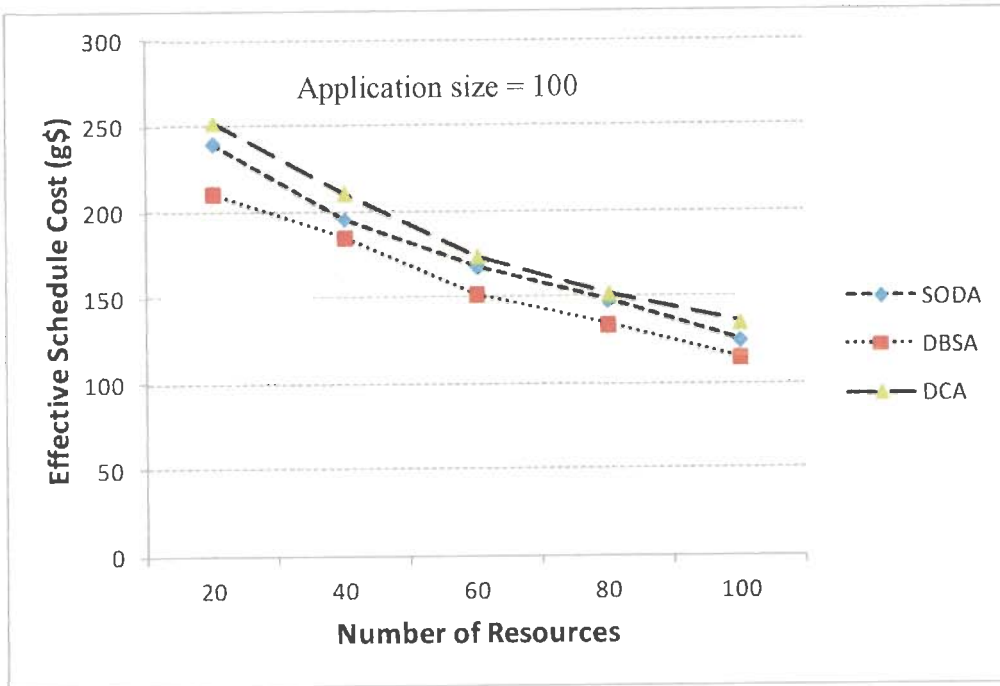
Figure 4.11: Gantt charts showing DBSA schedules (a) to (c) Primary scheduling; (d) Secondary scheduling with sliding constraints (+10% of makespan in primary scheduling)

In Figure 4.11(b), the above schedule is optimized after removing some duplications (n_1 on p_2, n_1 on p_4) whose removal keeps the makespan within maximum allowable limit. It reduces the economic cost of the schedule to 17.31 g\$ because of the removal of redundant duplications. Again, we identify those tasks or sub-schedules which have been duplicated over other

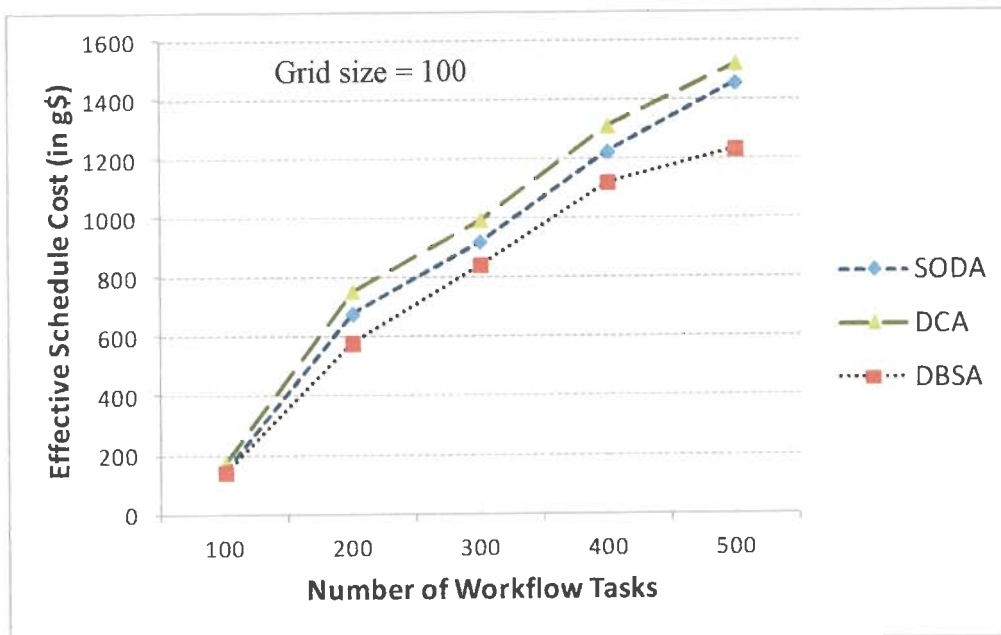
resources in primary scheduling. We try to remove them whose descendent tasks are receiving input data from their duplicated version. Such sub-schedules or tasks may be removed which reduces the economic cost of the schedule further to 14.32 g\$ as shown in Figure 4.11(c). The tasks in the above schedule are then tried to reschedule over cheaper resources if it reduces the economic cost while keeping the makespan within maximum allowable limit. The schedule in Figure 4.11(c) is modified in order to reschedule tasks from resource P3 to P4 which reduces the economic cost to 9.32 g\$ while makespan is kept below 18 (+10% of makespan in primary scheduling) yielding a schedule shown in Figure 4.11(d). The real scenarios of schedules shown in Figure 4.11 are obtained through simulation and presented in Figure 4.13. Full calculations for this example have been explained in Appendix – B.

4.3.5 Performance Comparisons and Result Analysis

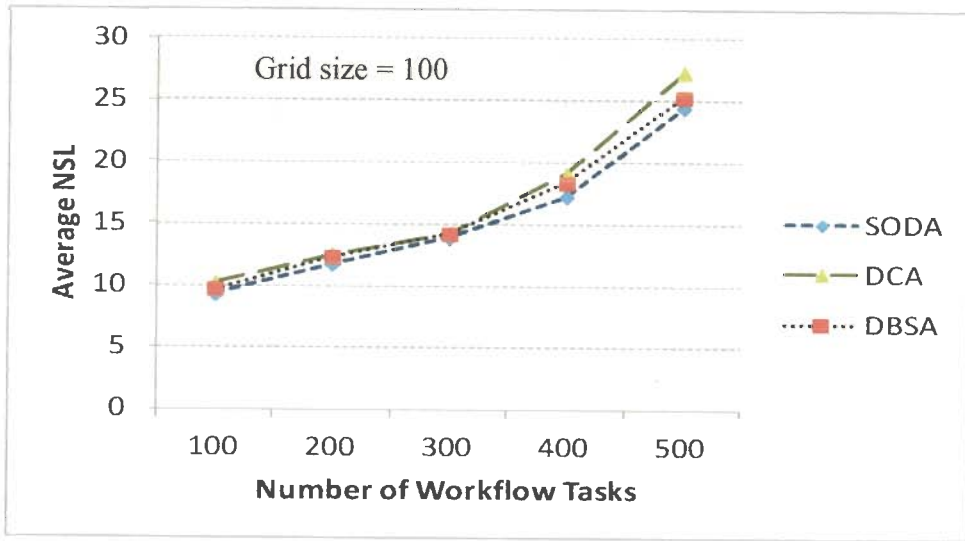
The pseudo code of DBSA for primary scheduling is same as in SODA (see Figure 4.4). The pseudo code for secondary scheduling is presented in Figure 4.10. The DBSA is implemented in the simulated grid environments for the evaluation of different graph sizes (100, 200, 300, 400 and 500) with different parallelism i.e., maximum out degree of nodes in the DAG (2, 4, 6, 8 and 10). The DBSA has been tested in the Grid of heterogeneous resources (20, 40, 60, 80 and 100). The DBSA has been compared with SODA and DCA. The results are analyzed for the performance metrics, i.e., effective schedule cost (ESC) and normalized schedule length. The simulation has been implemented in MATLAB using scheduling toolbox (see Appendix – A).



(a)



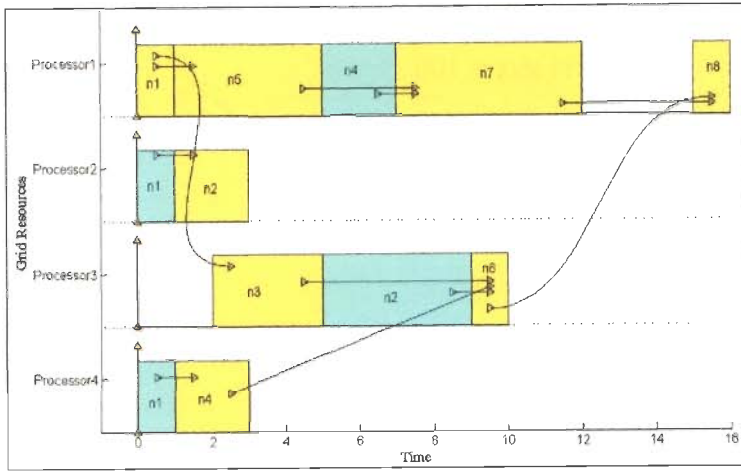
(b)



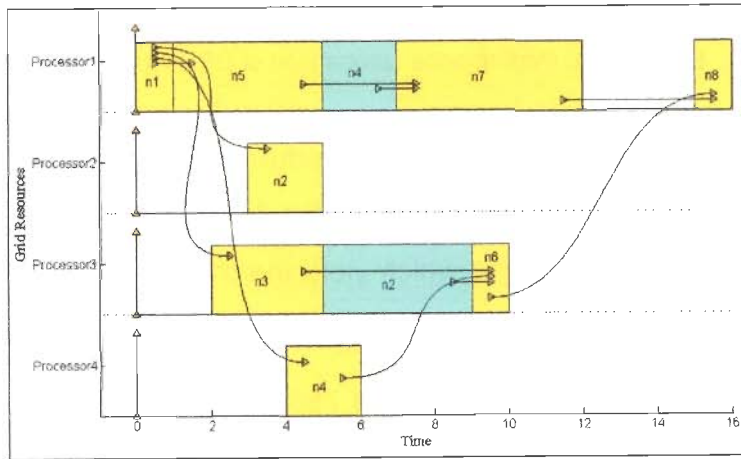
(c)

Figure 4.12: Performance comparison of DBSA algorithm

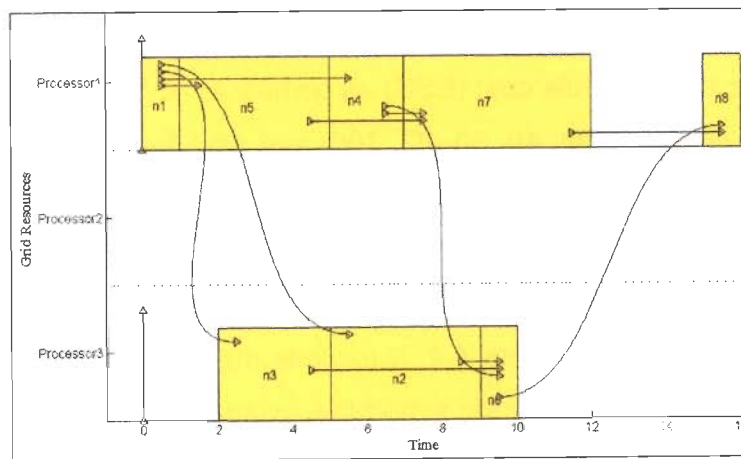
The DBSA is run for primary scheduling for both the criteria (i.e., makespan and economic cost) to find the best and worst solutions for primary criteria (c_1^{best} and c_1^{worst}) which yield the maximum sliding constraint, i.e., the difference $|c_1^{worst} - c_1^{best}|$. The algorithms are run for three different sliding constraint values: 25%, 50% and 75% of $|c_1^{worst} - c_1^{best}|$. The simulated results and graphs reveal that the proposed bi-criteria scheduling approach outperforms the SODA and DCA algorithm in terms of both economic cost and schedule length. In Figures 4.12(a) and 4.12(b), the DBSA yields the reduced effective schedule cost (ESC) as compared to SODA and DCA w.r.t. different grid size (i.e., 20, 40, 60, 80, 100) and workflow size (i.e., 100, 200, 300, 400, 500) while Figure 4.12(c) illustrates that DBSA generates longer schedules as compared to SODA due to sliding constraints in makespan. The simulation parameters for modeling workflows and Grid computing system are presented in Table 4.4. It exploits duplications to obtain shorter schedules and then improves the schedules using sliding constraints in order to remove useless and unproductive duplications to minimize economic cost. This, DBSA is able to be adopted for scheduling workflow applications in grid computing environments.



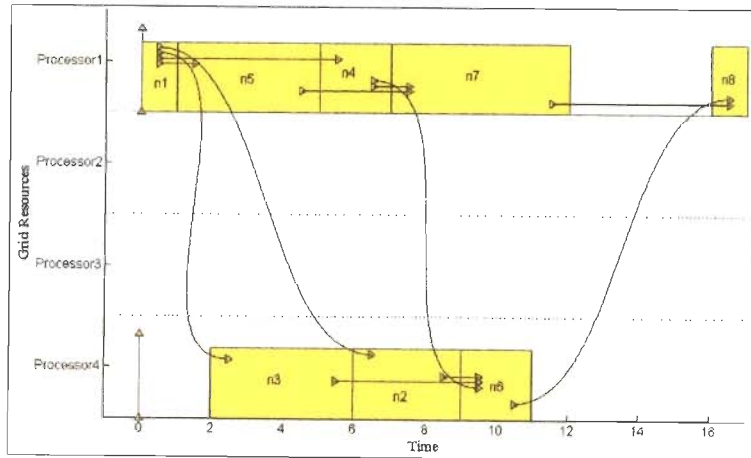
(a) Primary schedule by duplication-based scheduling heuristic



(b) Modified primary schedule after removal of useless duplications



(c) Modified primary schedule after removal of unproductive sub-schedules



(d) A secondary schedule after sliding makespan (+10%<18) to minimize economic cost

Figure 4.13: Real scenario of schedules of Figure 4.11 generated using Grid scheduling tool

4.4 Summary and Discussions

The work presented in SODA and DBSA shows that such multi criteria approach is quite beneficial in scheduling of workflow applications in grid environments. To exploit the grid resources without increasing scheduling time and cost, duplication approach significantly improve schedules. Loss approach, used to minimize the execution cost, is very graceful if the effect on makespan after migrating task to cheaper resource is minimal. The overall worst case computational complexity of these algorithms (SODA and DBSA) comes out to be $O(pn^2d_{max} + e)$ where d_{max} is the maximum in/out degree of a task in application DAG. In the next chapter, we are working to extend our algorithms by applying different QoS constraints posed in heterogeneous Grid environments. Also, in future we intended to implement and test the performance of our algorithm in real world Grid environment.

5.1 Overview

Single criteria and multiple criteria workflow scheduling heuristics have been discussed in previous chapters. They optimize execution time and monetary cost (economic cost) using duplication based approach. Quality of Service (QoS) is a concern in many grid applications. In many situations, user's tasks may need more information such as QoS levels of available resources and associated costs before making decisions. Furthermore, QoS levels and prices offered by service providers may be highly diverse and may not be directly correlated with the utility perceived by the users. For example, users may prefer some assignments which have slightly longer execution times but offer large savings in execution cost. Workflows have been recognized as an important application for Grid systems. However, composing and deploying such workflows on dynamic and heterogeneous distributed Grid resources to meet users' QoS requirements is a challenging task. Most parts of current research are concentrating on how the QoS requirements affect the resource assignment and then the performance of the other parts of the applications.

This chapter has been addressed to introduce two new scheduling heuristics (Trust-MOS, AQuA) for QoS requirements of tasks such as Trustworthiness and Availability of grid resources.

5.2 Trust Oriented Multi Criteria Scheduling

5.2.1 Preamble

Trust has been recognized as an important factor for task scheduling in Grid environments. Managing trust is crucial in a dynamic grid environment where grid nodes and users keep joining and leaving the system. Considering trust in comparatively open structures like Grids is a very relevant topic. Although, all computational Grids and their toolboxes include certain secure techniques such as GSI [22] in Globus which supports single-sign credentials and the collaboration between local secure strategy and that of whole system, there is a need to obtain a trustworthy resource as the resources in Grid are inevitably unreliable and unsafe. Therefore, there must be a mechanism to evaluate and manage the trust levels of Grid nodes while scheduling the tasks. Security considerations may increase the cost of executing applications in Grid. The objective of the current security solutions is to provide the Grid system participants with certain degree of trust so that their resource provider nodes or user programs will be secure.

The resource sharing in Grid may lead to the illegal users acquiring much higher trust level to access the resources that they have no right to access. These illegal users may destroy certain resources and their information for ever. This chapter addresses the *Grid Trust Model* to get solutions for these problems. A reliable trust mechanism is applied for management of dynamic trust level of resources based on a trust function. This trust model is integrated with *Grid Resource Management and Scheduling Model* to obtain more realistic schedule. Scheduling of tasks in grid with the characteristics of dynamism,

heterogeneity, distribution, openness, voluntariness, uncertainty, and deception, is a complex optimization problem and several different criteria are needed to be considered simultaneously to obtain a realistic schedule. Usually, execution time and economic cost (cost of executing a task on grid resource) are taken important criteria [124, 139].

A majority of problems in the literature address a single criterion scheduling [116, 124]. Considering multiple criteria with trust-oriented scheduling enables the decision maker to propose a better solution. Users in the areas such as in national intelligent analysis, banking, and financial data analysis etc will be greatly benefitted by trust oriented management and scheduling of grid resources. Traditionally, to cope up with these security issues, several methods such as to encrypt the data of execution and analysis, or isolate them from the Internet, and then schedule them on local resources have been used. Therefore, an efficient scheduling algorithm is needed to execute tasks on trustworthy resources while assuring the high speed of communication, reduce the task execution time and economic cost, lower the ratio of task failure execution, and improve the security of important data execution.

In literature, the problem of multi-objective scheduling considers the execution time (or makespan) and the economic cost as two independent criteria [130]. But, very few research efforts have been done towards trust-oriented approach in multi-objective workflow scheduling. Existing scheduling algorithms largely ignore the security induced risks involved in dispatching tasks to remote sites. In [130], a bi-criterion scheduling algorithm (DCA) based on sliding constraints has been introduced for workflow applications in grid environments. But, it does not address the trust-oriented issues of Grid resources. In [145], Zhu et al. proposed a *Grid Economic Model* in which they presented two scheduling algorithms for independent tasks: trust-aware time optimization scheduling algorithm within budget constraints, and trust-aware

cost optimization scheduling algorithm within deadline constraints. But, they do not address the scheduling for workflows.

In [6], a trust-aware model between the resource producers and consumers has been proposed. In [2], a model based on experience and reputation for supporting trust has been proposed. This model allows entities to decide which other entities are trustworthy. Song et al. [116] enhanced the Min-Min and Suffrage heuristics and proposed a novel space-time genetic algorithm for trusted job scheduling. Abawayj [1] presented a *Distributed Fault-Tolerant Scheduling (DFTS)* algorithm to provide fault-tolerance to task execution in Grid systems. But, they do not focus on multi-criteria approach in scheduling.

This chapter presents a trust-based multi-criteria optimization model for task scheduling in grid environment. Three criteria namely execution time, economic cost, and trustworthiness are evaluated to compute the quality of the schedule for workflow applications. Trustworthiness is an indicator of the quality of the resource's service. It is often used to predict the future behavior of the resource. Intuitively, if a resource is trustworthy, it is likely that it will provide good services in future transactions too. The scheduler must carefully evaluate the compromise involved in considering multiple criteria in scheduling applications. In this research work, a new hybrid multi-objective scheduling strategy is used to evaluate a trust-oriented time and cost optimized schedule in Grid. The important issue for executing an application in the Grid is how to obtain trustworthy resource for the task to be executed over it.

A trust-based decision in a specific domain is a multi-stage process. The first stage consists of identifying and selecting the proper input data, i.e. the trust evidences. In general, these are domain-specific and they result from an analysis conducted over the application involved. In the next stage, a computation is performed over evidences to produce trust values i.e., the estimation of the trustworthiness of resources in that particular domain. The selection of evidences and the subsequent trust computation are informed by a

notion of trust, defined in the *Grid Trust Model* (Section 5.2.2). Finally, the trust decision is taken by considering the computed values and exogenous factors, like disposition or risk assessments.

5.2.2 Trust Model for Grids

As definition [31, 79, 118, 145]: “Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subject to the entity’s behavior and applies only within a specific context at a given time”. Trust value of any resource is computed on past experiences of resources in a specific context. There are several issues that arise in a real grid environment. First, malicious nodes, which may damage resources or act against a protocol and try to attack the Grid system, can degrade the performance of system drastically [31]. Second, selfish nodes or free riders, which may consume but do not contribute resources, have been a serious issue [4].

The trust life cycle is composed of three different phases: trust formation phase, trust negotiation phase, and trust evaluation phase. The *trust formation phase* is done before any trusted group is formed. It contains mechanism to develop trust functions and policies. The *trust negotiation phase* is activated when a new untrusted system joins the current distributed system. The *trust evaluation phase* reevaluates and updates the trust values based on transactions performed in the system. In this work, a trust model based on Eigen-Trust Model [64, 118] is used to distinguish trustable nodes from malicious nodes in the grid environment. In Grid, local trust values of nodes are assigned by other nodes after each transaction. For example, when resource p_i executes a task after receiving data from resource p_j , it may rate the transaction as successful $x_{ij} = 1$, or unsuccessful $x_{ij} = -1$. The local trust value

(ltv_{ij}) can be defined as the sum of the ratings of each transaction that node p_i has executed tasks from node p_j .

$$ltv_{ij} = \sum x_{ij} \quad (5.1)$$

The system should select a very few number of pre-trusted nodes. In order to aggregate local trust values, it is essential to normalize them. Otherwise, malicious nodes may obtain arbitrarily high local trust values from other malicious nodes, and assign arbitrarily low local trust values to good quality nodes, easily subverting the system. We can define a *normalized local trust value* v_{ij} , as follows:

$$v_{ij} = \begin{cases} \frac{\max(ltv_{ij}, 0)}{\sum_l \max(ltv_{il}, 0)} & , \text{ if } \sum_l \max(ltv_{il}, 0) \neq 0 \\ r_j & , \text{ otherwise} \end{cases} \quad (5.2)$$

Here, r_j can be defined as:

$$r_j = \begin{cases} \frac{1}{|R|} & , \quad j \in R \\ 0 & , \text{ otherwise} \end{cases} \quad (5.3)$$

where R is a set containing pre-trusted nodes. Now, we wish to aggregate the normalized local trust values. In Grid, node p_i could get recommendations from its acquaintances about other nodes:

$$t_{ij} = \sum_k v_{ik} v_{kj} \quad (5.4)$$

where t_{ij} represents the trust that node p_i places on node p_j based on asking its friends. If we write Equation (5.4) in matrix notation, then we obtain a trust vector $\vec{t}_i = V^T \vec{v}_i$, where V represents the matrix $[v_{ij}]$, $\vec{v}_i = [v_{i1}, v_{i2}, \dots, v_{in}]^T$ contains v_{ij} , and $\vec{t}_i = [t_{i1}, t_{i2}, \dots, t_{in}]^T$ contains t_{ij} . To gain a wider view, node p_i may wish to ask its friends' friends, then $\vec{t}_i = (V^T)^2 \vec{v}_i$. If node p_i continues in this manner ($\vec{t}_i = (V^T)^n \vec{v}_i$), where V is irreducible and aperiodic, it will have a complete view of the global Grid environment after n large iterations.

In Grid, there is a potential for malicious collectives to form. A malicious collective is a group of malicious nodes who know each other, who give each other high local trust values and give other nodes low local trust values in order to subvert the system order and acquire high global trust values. This problem can be resolved by taking:

$$\vec{t}^{k+1} = (1 - \varphi)V^T \vec{t}^{(k)} + \varphi \vec{r} \quad (5.5)$$

where φ is a constant between 0 and 1. This is a way to break collective by having each node place at least some trust on the accessible nodes in Grid that are not the parts of collective. Here, it is important that no pre-trusted node should be a member of malicious collective. In the Grid environment, V and \vec{t} are stored in each node and node p_i can compute its own global trust value t_i as follows:

$$t_i^{(k+1)} = (1 - \varphi)(v_{1i} t_1^{(k)} + v_{2i} t_2^{(k)} \dots + v_{ni} t_n^{(k)}) + \varphi r_i \quad (5.6)$$

The global trust value (t_i) of each resource is computed using Equation (5.6) and updated dynamically. The algorithm to compute t_i is shown in Figure 5.1.

Algorithm 5.1: Eigen-Trust

```
Initial:  $r_i$  the initial trust value of node  $i$ 
Eigen-Trust()
1. for each node  $l$  do {
2.   for all nodes  $j$ , where  $j \neq i$ ,  $t_j^{(0)} = r_j$ 
3.   repeat
4.     compute  $t_j^{(k+1)} = (1 - \varphi)(c_{1j}t_1^{(k)} + \dots + c_{mj}t_m^{(k)}) + \varphi.r_j$ 
5.     send  $c_{lj}t_j^{(k+1)}$  to all nodes
6.     Wait for all nodes return  $c_{jl}t_j^{(k+1)}$ 
7.   until  $|t_j^{(k+1)} - t_j^{(k)}| < \varepsilon$ 
8.   endfor
9. endfor
```

Figure 5.1: Eigen-Trust algorithm

In [64], Kamvar et al. define a probability ε , known as the *teleport probability* (the probability of visiting random nodes) for historical reasons, which measures how much trust the pre-trusted nodes receive due to their pre-trusted status. The default value of ε is 0.15.

5.2.3 Performance Metrics

A workflow application model, described in section 3.2.3, is used in this scheduling approach. The Grid resource model presented in section 3.2.2 has been used to simulate the proposed trust oriented multi-objective scheduling approach. A fairly static methodology has been used for defining the weights of the computational tasks and communicating edges in the DAG. In scheduling, a workflow of tasks is submitted to the Grid meta-scheduler where tasks are queued in non-decreasing order of their b-levels.

The economic cost (EC) is the summation of the economic costs of all workflow tasks scheduled on different resources which can be computed as:

$$EC = \sum_{j=1}^p \sum_{i=1}^n D_{ij} \quad (5.7)$$

where D_{ij} is the economic cost of executing task n_i onto trusted resource p_j . It can be computed as:

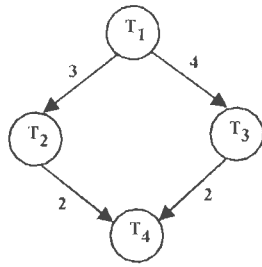
$$D_{ij} = M_j \times \alpha(p_j) \times \omega_{ij} \quad (5.8)$$

where M_j is the per MIPS machine cost (in grid Dollar or g\$) of executing task on resource p_j and ω_{ij} is the execution time that task n_i takes to run on resource p_j . In this research, trustworthiness [16] of grid resources has been used as performance metrics for comparing trust-oriented multi-objective scheduling heuristic (Trust-MOS) with DCA algorithm. It is an average of the global trust values of the resources used in scheduling a workflow.

The simulated analysis and results reveal that trust oriented approach strengthens the multi-objective scheduling problems yielding better make span and reducing the economic cost. Another performance metrics is *trust cost overhead* (TC) which can be computed using Equation (5.10) for the tasks scheduled on different resources in the Grid. It is found that Trust-MOS generates less trust cost overhead as compared to DCA algorithm and reduces the execution time and economic cost overhead subsequently in multi-objective optimization process. It reflects that the resources with higher global trust values reduce the task failure ratio and produces good quality schedules with lesser economic cost.

5.2.4 Trust–MOS Scheduling Algorithm

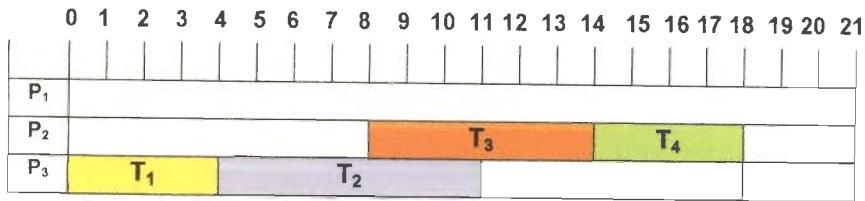
We have proposed an intelligent scheduling algorithm called trust-oriented multi-objective scheduling algorithm (Trust–MOS) to address the multi-objective optimization problem. The pseudo code of the proposed algorithm is described in Algorithm 5.2 and 5.3 (Figures 5.3 and 5.4). It is divided into two major phases: (1) Primary Scheduling – Optimization for the primary criterion considering trust, (2) Secondary Scheduling – Optimization for the secondary criterion while keeping the primary criterion within the defined sliding constraint limit. The total execution time (makespan) is selected as primary criterion while the economic cost is selected as secondary criterion. An efficient list-based scheduling heuristic [124] has been applied for execution time optimization. A cost optimization algorithm such as GreedyCost [139] has been used for selecting the cheapest resources for scheduling. We apply an efficient sliding constraint method to define the user requirements [130]. The user is expected to define “sliding constraint” for the primary criterion i.e., how much the final solution may differ from the best solution found for the primary criterion. Figure 5.2 illustrates the schedules produced by Trust-MOS and DCA algorithms for primary criteria i.e., makespan. It is clearly indicated in this example that Trust-MOS assigns the tasks to most trustworthy resources than DCA at the cost of execution time. The makespan is better in case of DCA but if we add the overhead of rescheduling if the resource is malicious than the actual makespan will improve in our case.



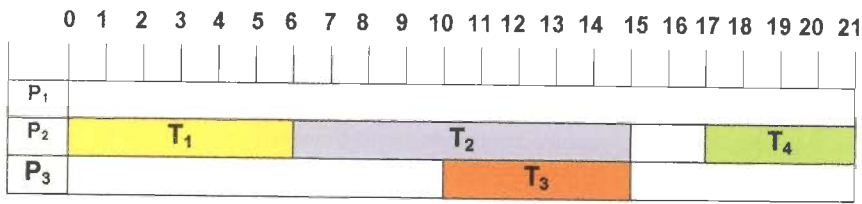
(a) A sample DAG

	P ₁	P ₂	P ₃
T ₁	5	6	4
T ₂	8	9	7
T ₃	7	6	5
T ₄	3	4	5
Trust Level	0.5	0.9	0.3

(b) Expected Time to Compute Matrix with Trust Levels



(c) Primary schedule of DCA algorithm (Trustworthiness=0.60, Makespan=18)



(d) Primary schedule of Trust-MOS algorithm (Trustworthiness=0.75, Makespan=21)

Figure 5.2: Relative schedules of DCA and Trust-MOS in primary scheduling

[A] Primary Scheduling

The objective of primary scheduling is to obtain the optimal schedule of workflow for the primary criterion (i.e., makespan) only while considering the trustworthiness of resources. It generates a preliminary solution with the total costs of the primary and the secondary criterions which can be denoted as c_1^{prel} and c_2^{prel} respectively. The task n_i is scheduled on to the resource p_j which minimizes the *trust-based finish time* (TFT_{ij}) for the primary criterion. It can be computed as:

$$TFT_{ij} = TC_{ij} + EFT_{ij} \quad (5.9)$$

where TC_{ij} is the trust cost overhead of assigning task n_i on resource p_j and EFT_{ij} is the *expected finish time* of task n_i on resource p_j . The *trust cost overhead* can be computed as:

$$TC_{ij} = (1 - t_j) \times \omega_{ij} \quad (5.10)$$

where t_j is the global trust value of resource p_j .

Algorithm 5.2: Primary Scheduling in Trust-MOS

1. Compute global trust value t_j for each resource p_j using **Eigen-Trust** Algorithm 5.1
2. Construct a priority based task sequence ξ based on higher b-level
3. *for* (each unscheduled task n_i in task sequence ξ)
4. *for* (each trusted resource p_j in the Grid)
5. Compute Trust Cost Overhead (TC_{ij}) using Equation (5.10)
6. Compute trust based finish time (TFT_{ij}) using Equation (5.9)
7. *end for*
8. Assign task n_i on resource p_j which minimizes TFT_{ij}
9. *end for*
10. Calculate execution time c_1^{prel} using Equation (5.12) and economic cost c_2^{prel} using Equation (5.7).

Figure 5.3: Primary scheduling algorithm of Trust-MOS

The pseudo code of the Algorithm 5.2 for primary scheduling (Figure 5.3) presents the primary scheduling based on trust oriented approach for primary criterion only. First, we compute the global trust values of available resources in Grid using the *Grid Trust Model* as described in section 5.2.2. These values are periodically updated. Then, we generate a priority based task sequence based on b-levels. Further, we compute the trust based finish time of each task on every resource and schedule task on to the resource which minimizes trust

based finish time. Finally, we compute the total execution time (c_1^{prel}) and total economic cost (c_2^{prel}) of the schedule generated for the primary criterion.

[B] Secondary Scheduling

The secondary scheduling optimizes primary solution for the secondary criterion (i.e., economic cost) while keeping the primary criterion within defined sliding constraint. It produces the best possible solution with the total costs of primary and secondary criteria which can be denoted as c_1^{final} and c_2^{final} . The sliding constraint is equal to L_T such that the primary criterion cost can be increased from c_1^{prel} to $c_1^{prel} + L_T$ in respect of reducing the secondary criteria cost in secondary scheduling. We assume that the maximum allowable execution time T_{Greedy} of workflow with cheapest economic cost C_{Greedy} can be computed using cost optimization algorithm GreedyCost [139]. Similarly, maximum allowable economic cost C_{Heft} of workflow with shortest execution time T_{Heft} can be computed using time optimization algorithm HEFT [124]. Thus, the sliding constraint for makespan can be computed as:

$$L_T^k = k \times (T_{Greedy} - T_{Heft}) \quad (5.11)$$

where k is the sliding constant. The value of k varies from 0.1 to 1.0 to provide us the sliding constraints that lie in ten equally distanced points for the makespan. In general, HEFT generated schedule contains shorter makespan as compared to GreedyCost algorithm. The makespan for the schedule may be computed as:

$$makespan = \max(TFT_{ij}) \quad \forall 1 \leq i \leq n \text{ and } 1 \leq j \leq p \quad (5.12)$$

The Algorithm 5.3 (Figure 5.4) presents secondary scheduling which optimizes both primary and secondary criteria. First, we compute the sliding constraints using Equation (5.11) that lie in ten equally distanced points for total execution time. This algorithm finds the alternative services (resources) for rescheduling already scheduled task such that makespan should not increase beyond the maximum allowable execution time limit ($c_1^{prel} + L_T$) and economic cost is lesser on these services. The task is rescheduled on to the resource which maximizes net_{profit} as specified in Figure 5.4. Finally, total execution time c_1^{final} and total economic cost c_2^{final} is estimated for optimized schedule re-generated for both criteria considering trustworthiness of grid resources.

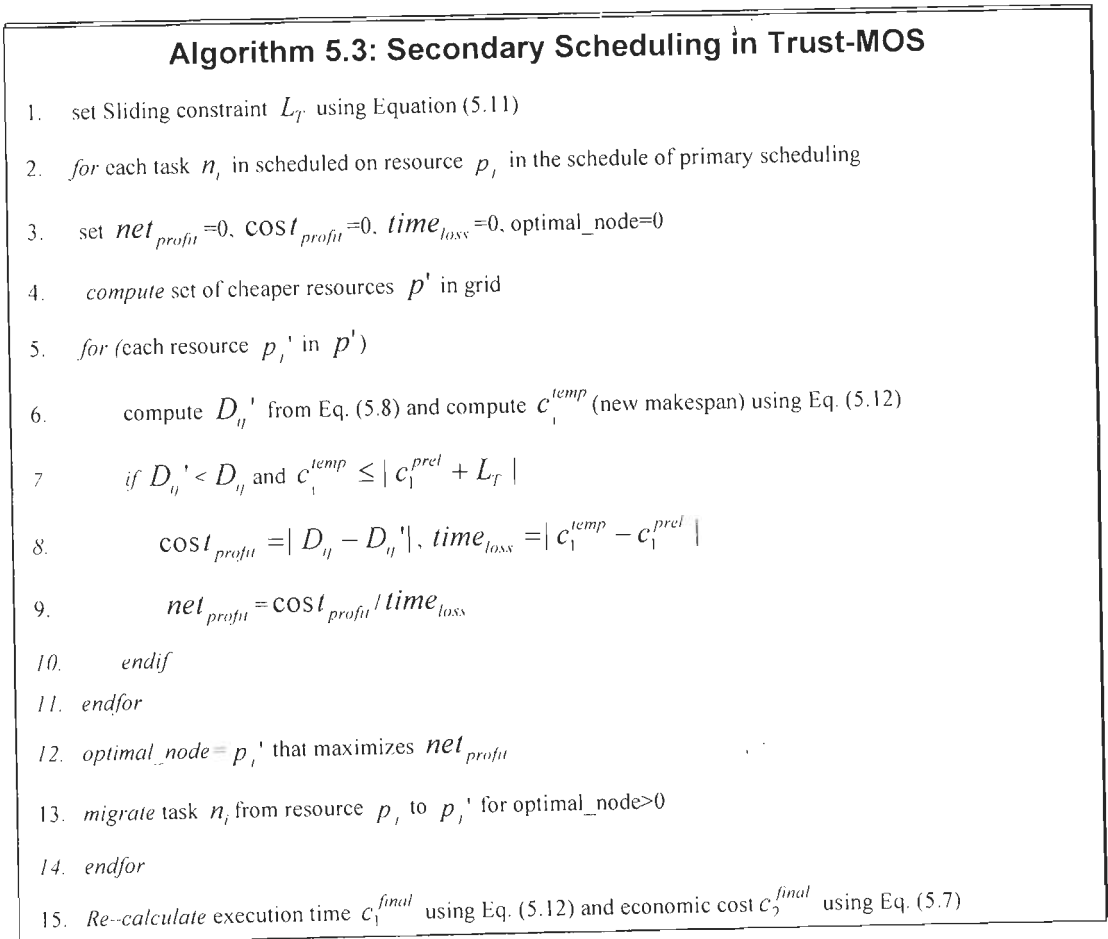


Figure 5.4: Secondary scheduling algorithm of Trust-MOS

5.2.5 Performance Comparisons and Result Analysis

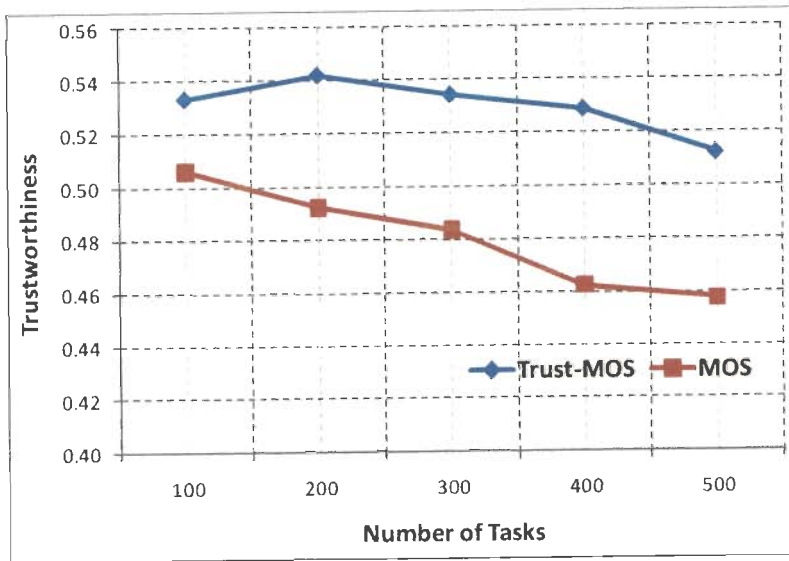
The algorithms shown in section 5.2.4 have been evaluated and validated with the help of simulated grid scheduling tool generated by us to run on a variety of random workflows. The experiments have been implemented for randomly generated DAGs on simulated grids of different sizes. The specification layout for simulation is given in Table 5.1. The proposed algorithm (Trust-MOS) has been executed and compared with bi-criteria scheduling algorithm i.e., Dynamic Constraint Algorithm (DCA) [130] for different workflows in Grids of different sizes. The DCA algorithm does not focus on trust related issues of resources in the Grid. Both the algorithms have been run under the same conditions for fair comparison i.e., each algorithm is run for each workflow to find best possible second criteria cost while keeping the primary criterion within the defined sliding constraint in the same grid environment.

Table 5.1: Grid environment layouts

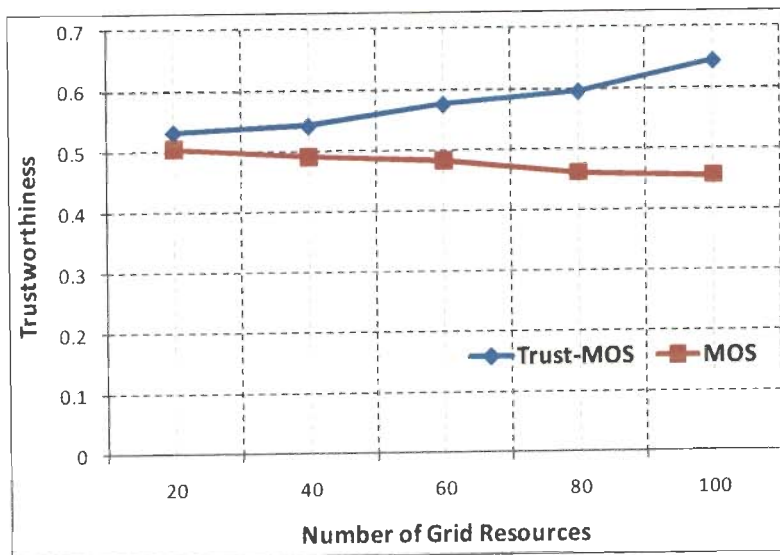
Number of Grid Resources	[20, 100]
Resource Bandwidth	[100 Mbps, 1 Gbps]
Number of Tasks	[100, 500]
Computation Cost of Tasks	[5 msec, 200 msec]
Data Transfer Size	[20 Kbytes, 2 Mbytes]
Resource Capacity (MIPS)	[220, 580]
Execution Cost (per MIPS)	[1 – 5 grid dollar per MIPS]
Sliding Constant	[0.1 – 1.0]

The Algorithm 5.2 (Figure 5.3) has been run for primary scheduling for both the criteria (i.e., makespan and economic cost) to find the best and worst solutions (assumed) for primary criterion (T_{Heft} and T_{Greedy}) which yield the maximum sliding constraint i.e. the difference $|T_{Greedy} - T_{Heft}|$. The Algorithm 5.3 (Figure 5.4) is run for ten equally spaced different sliding constraint values to obtain the optimized execution time and economic cost. In Figure 5.5(a) and Figure 5.5(b), Trust-MOS algorithm excels the DCA algorithm in terms of trustworthiness of Grid resources used in executing the given workflow.

Trustworthiness of both algorithms has been compared using workflows of different sizes (100, 200, 300, 400 and 500) as depicted in Figure 5.5(a), and using grids of different sizes (20, 40, 60, 80 and 100) as depicted in Figure 5.5(b). In Figure 5.6(a), the comparison has been illustrated for trustworthiness with respect to different sliding constants ranging from 0.1 to 1.0 which produces the ten equally spaced sliding constraints for primary criterion cost.

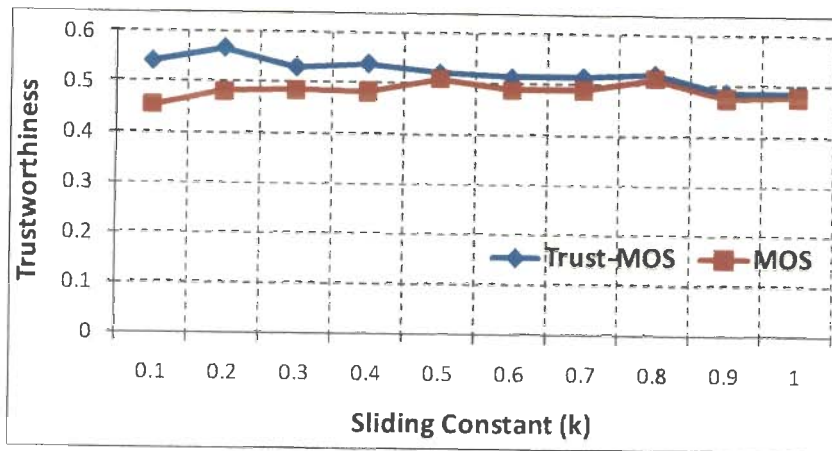


(a)

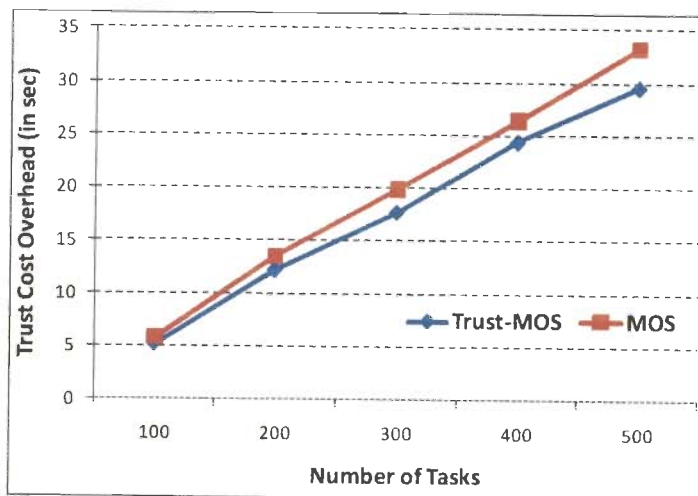


(b)

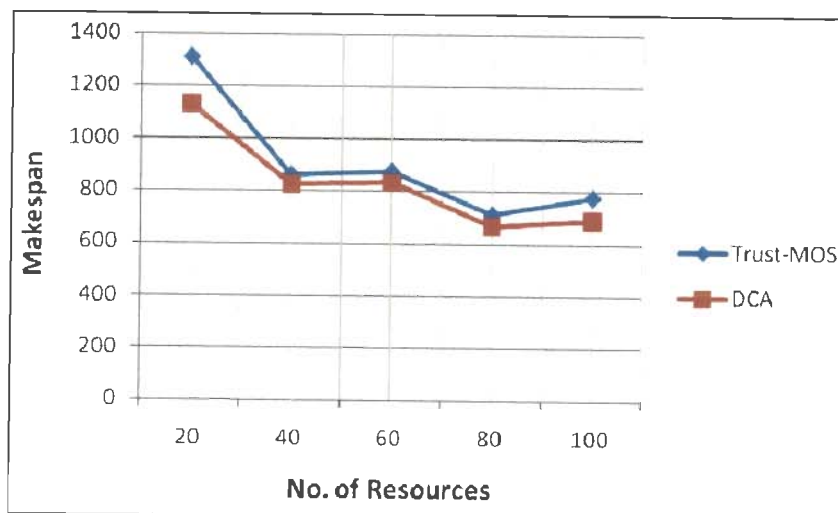
Figure 5.5: Comparison of trustworthiness w.r.t. (a) Number of tasks; (b) Number of grid resources



(a)



(b)



(c)

Figure 5.6: (a) Comparison of trustworthiness for different sliding constants; (b) Comparison of trust cost overhead w.r.t. number of tasks; (c) Relative makespan w.r.t. number of resources

In this research, an intelligent trust-based multi-objective workflow scheduling approach has been presented and analyzed. To the best of our knowledge, this is the first trust-based multi criteria algorithm for workflow scheduling in Grid environment. The algorithm has been implemented to schedule different random DAGs onto the Grids of different sizes. Different variants of the algorithm were modeled and evaluated.

5.3 Availability Aware QoS Oriented Scheduling

A major challenge in task scheduling is the availability of resources as they are highly dynamic, heterogeneous and unpredictable in Grid computing systems. The task-resource mapping in such a non-deterministic computing environment leads to concern over Quality-of-Service (QoS) constraints imposed by application tasks to identify capable resources. To achieve a better makespan is a key issue as Grid resources exhibit different capabilities and are not continuously available for computation. In this section, resource availability and bandwidth have been considered as QoS parameters of Grid resources. We modify the QoS guided Min-Min (QGMM) heuristic [60] to introduce a multidimensional QoS scheduling strategy called Availability aware QoS oriented Algorithm (AQuA) for task scheduling in grids to fulfill the availability and bandwidth requirements of application tasks. This approach gives priority to high QoS tasks to get scheduled first to meet the QoS tasks as much as possible. The time and accuracy of various computing and data-intensive tasks are limited by the availability of the resources within the silos. The proposed strategy is able to prioritize the utilization of highly available resources in the Grids and therefore increases the availability of grid computing systems to successfully execute applications without adversely affecting the makespan.

5.3.1 Preamble

In grid, resources exhibit different availability properties and patterns over time due to administrative policies belonging to different domains. There exist many high performance applications like healthcare applications, finance management applications, military applications etc which may require resources with high availability since severe damage or fatal errors could occur when even only one computing resource becomes unavailable.

The basic assumption in scheduling theory is that resources are always available for computation [105, 110]. This assumption might be reasonable in few cases, but it is not valid in grid scenario where resources are not continuously available due to several reasons such as maintenance requirements, breakdowns, administrative policies or other constraints. Availability can be defined as the total time a computing resource is functional during a given interval. A large number of grid resources may be unavailable at any time due to wide range of policies for when and how to make these resources available to the Grid. These policies are based on resource usage patterns and owner's other preferences, like shutting down the resources during night. To obtain a highly available resource is an issue as resource is inevitably unreliable and dynamic in Grid computing environment. Therefore, there must be a mechanism to evaluate and manage the availability levels of grid resources while scheduling the tasks.

In this research, task scheduling focuses on a set of independent tasks with QoS constraints. For example, some critical tasks may require bandwidth not less than 1.0 Giga bits/sec and availability of computing resources should be greater than 90% for their execution. The Grid Scheduler needs to consider the QoS requirements of application tasks to discover a set of potential resources that meets QoS requirements. Then, application tasks are assigned over grid resources in order to get a better match between tasks and resources which meets predefined scheduling criteria such as minimizing the makespan.

The meaning of QoS is highly dependent on particular applications, from hardware capacity to software existence. Usually, QoS is a constraint imposed on the scheduling process instead of the final objective function [10, 77].

To address the challenges of non-dedicated computing and network resources in the Grid, we adopt the prediction models [89, 119]. A newly proposed long term application level prediction model [53] is exploited to predict the task execution time for a task/resource pair in a dynamic running environment for large applications. In this work, we embedded the QoS information into the scheduling algorithm to make a better match among different levels of QoS request/ supply. The grid schedulers and resource brokers need information about resource availability properties and predictions about their future availability.

Most conventional scheduling algorithms have concentrated only on high throughput with the goal of reducing makespan, completely ignoring the availability and bandwidth requirements of tasks. To obtain a resource with high availability is an open issue in the Grid as the resources are inevitably unreliable and dynamic [46]. This motivates us to adopt the availability and bandwidth constraints of tasks while scheduling in grid scenario. Our scheduling model addresses the match of QoS request from application to the QoS provided by the grid resources. The involvement of QoS has an effect on the resource selection step in the scheduling which influences the final objective function.

5.3.2 Related Work

QoS resource management and scheduling is an important component of the Grid. In literature [20, 23, 24, 85], many heuristics have been proposed to obtain the optimal match between application tasks and resources. The scheduling is performed in multiple steps to solve the problem of matching

application requirements with resource availability while providing quality of service. It is challenging to achieve high throughput and high availability requirements simultaneously, because they are two conflicting objectives. For example, it is unacceptable to assign a critical task with high availability requirements to a computing resource that offers high speed but low availability. An efficient scheduling scheme is desirable to guarantee the tasks' availability requirements while reducing the completion time.

A task allocation scheme is investigated for scheduling tasks with availability constraints [110]. In [12, 13, 127], a Multi-Resource Scheduling (MRS) algorithm has been proposed. The main objective of this algorithm is to obtain a minimal execution schedule through efficient management of available Grid resources. They introduce the concept of a two dimensional virtual map and resource potential using a co-ordinate based system. To further develop this concept, a third dimension was added to include resource availabilities in the Grid environment. Smith [115] introduced a mathematical model for resource availability and proposed a method to maintain availability information as new reservations or assignments are made. Lee [75] addressed two machine scheduling problem in which an availability constraint is imposed on one machine as well as on both machines. A dynamic programming strategy for parallel machine scheduling problem with availability constraints is proposed. He et al. [60] presented a QoS guided Min–Min (QGMM) heuristic for independent task scheduling problems, but it does not consider the availability requirements imposed by application tasks for scheduling. Foster et al. [45] presented general purpose architecture for reservation and allocation (GARA) that supports flow–specific QoS specification, immediate and advance reservation, and online monitoring and control of both individual resources and heterogeneous resource ensembles.

Dogan and Ozguner [38] considered the problem of scheduling a set of independent tasks with multiple QoS requirements, which may include timeliness, reliability, security, version, and priority in a grid in which resource

prices can vary with time during scheduling time intervals. Golconda and Ozguner [52] compared five QoS-based scheduling heuristic from the literature in terms of three performance parameters, namely number of satisfied users, makespan, and total utility of meta-task.

The main contribution of this research is to consider QoS constraints (i.e., availability of computing nodes and bandwidth of network resources) of applications while selecting the suitable resources in the grid environments. The experimental analysis and results show that the proposed algorithm (AQuA) is able to utilize the highly available resources in grid and therefore increases the reliability to successfully execute applications without adversely affecting the makespan.

Embedding multi-dimensional QoS into task scheduling is an open problem and an issue which has been addressed in this work. Availability and bandwidth are two QoS constraints in resource selection for successful execution of tasks. The primary objective of any scheduling problem is to obtain a schedule with minimum execution time (or makespan). The makespan is the maximum completion time i.e., the maximum between the start time of the first task and the finish time of the last task in a task set scheduled on different grid resources. In independent task scheduling, the makespan of the complete schedule is defined as $\max_{t_i \in T} (CT_i)$ and it is a measure of throughput. The objective of grid scheduling is to increase availability of grid computing systems while meeting the QoS requirements of tasks without affecting the makespan adversely.

5.3.3 Grid Resource Model

Most existing scheduling heuristics developed for distributed computing systems do not factor in availability requirements imposed by application tasks. To explore this issue, each computing node in grid is modeled using nodes'

computing capacity and availability. A Grid system can be modeled as a finite set of heterogeneous computing resources as:

$$R = \{r_1, r_2, \dots, r_n\} \quad (5.13)$$

Each resource is characterized by its bandwidth capacity and availability level. These resources are not entirely dedicated to the Grid. Based on resource availability policies, we can classify them in three main categories as in Austrian Grid [121]: (1) the dedicated resources, which are meant to be always available to grid users; (2) the non-dedicated resources, which are available in grid as long as they are turned on; and (3) the on-demand resources, which are made available to grid only on demand from the users for executing large scale applications or experiments. This classification in turn reflects the availability of the resources.

1. *Availability*. It can be defined as the total time a computing resource is functional during a given interval. It is a probability that the system is continuously performing at any random period of time. It can be computed as:

$$availability = \frac{\Delta\tau}{\tau} \quad \text{where } 0 \leq \Delta\tau \leq \tau \quad (5.14)$$

where $\Delta\tau$ is the time during which a resource is functional and τ is total time of monitoring the availability of resource.

2. *Bandwidth*. It is the network capacity (in bits per second) to access a computational resource for data transfer.

The availability and capacity of grid resources, e.g. number of hosts and network links, fluctuates. Thus, to determine a priori the completion time of a task on a particular resource is difficult. Moreover, a task may fail to complete the execution due to failure of the resource or network link. We assume that an entity (or distributed entities) exists that computes and provides prediction information of resources' availability and links' bandwidth in the grid. If the resource behavior follows a certain distribution, stochastic process analysis can be used to predict the future resource measurements on a fixed time point or during a certain interval of time. In case of network bandwidth, we consider the dynamically forecasted prediction information by facilities such as the Network Weather Service (NWS) [131]. NWS is an agent system deployed on the Grid to periodically monitor resource performance. Usually, NWS monitors certain performance measurement in 10 second intervals.

The objective of the proposed scheduling algorithm is to obtain a good task allocation decision for application tasks to satisfy their multidimensional QoS requirements and maintain an ideal performance of makespan.

5.3.4 Grid Application Model

An application is modeled as a set of independent tasks (or meta-task) as follows:

$$T = \{n_1, n_2, \dots, n_m\} \tag{5.15}$$

The expected execution time for each task on different resource is known prior to task execution which can be represented by a *ETC* (expected time to compute) matrix. The above prediction is true for dedicated resources where availability of resources is 100% (i.e., 1). But, in practical, this assumption may not be true as grid resources are highly dynamic and

unpredictable. Therefore, the ETC of task n_i can be modified to add availability overhead.

$$ETC_{ij}' = \frac{ETC_{ij}}{avail(r_j)} \quad (5.16)$$

where $avail(r_j)$ is the availability level of resource r_j . The completion time of task n_i on resource r_j can be computed as:

$$CT_{ij} = AT(r_j) + ETC_{ij}' \quad (5.17)$$

where $AT(r_j)$ is the earliest time when resource r_j is available to take up task n_i . Initially, $AT(r_j) = 0$ for all resources $r_j \in R_j$.

5.3.5 AQuA Scheduling Algorithm

QoS-based Grid scheduling system architecture is depicted in Figure 5.7. In this model, users' tasks with QoS constraints are submitted to the grid scheduler that builds a set of potential resources for each task meet its QoS request and then, schedule the tasks over these resources using Min-Min approach [60]. A task set T can be divided into two task sets T_1 and T_2 such that T_1 contains m_1 tasks with QoS requirements and T_2 contains m_2 tasks without QoS requirements respectively. It is assumed that there exist few tasks (or critical tasks) with QoS requirements in an application. The tasks in task set T_1 get priority in scheduling over those in T_2 . For every task, a resource from a set of potential resources is selected that provides the minimum completion time. The task with minimum MCT is selected to be scheduled first and so on.

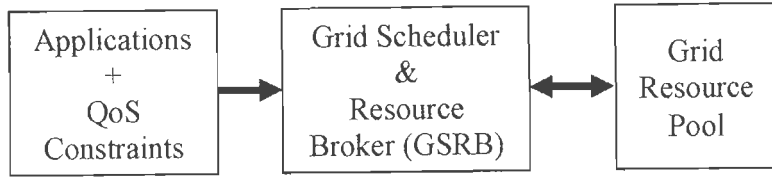


Figure 5.7: QoS based Grid scheduling system

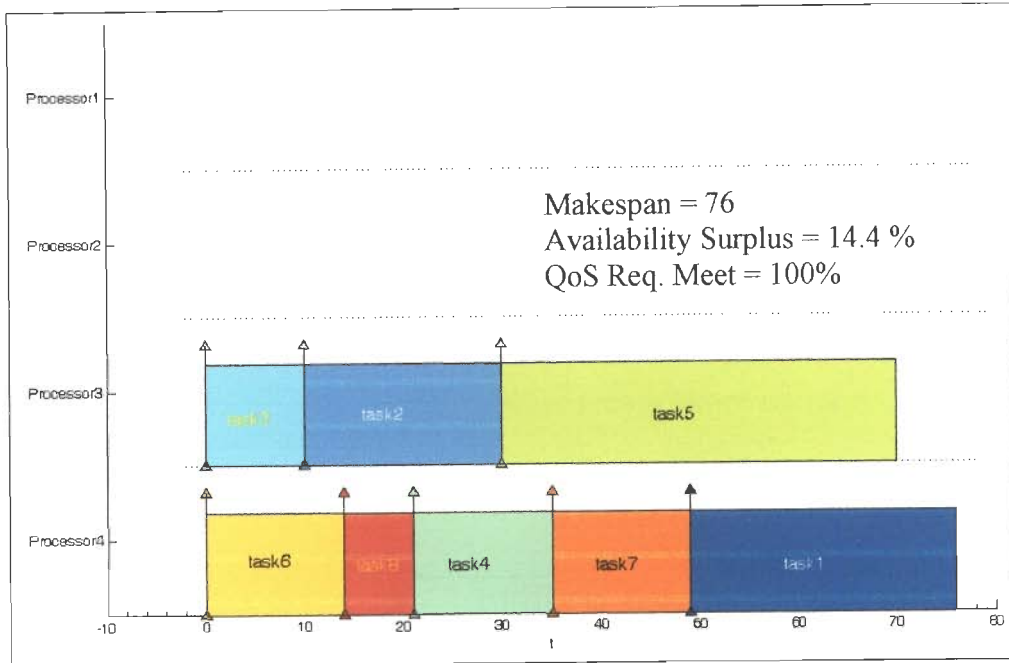


Figure 5.8: Gantt chart showing schedule of AQuA

The pseudo code of the proposed scheduling algorithm (AQuA) is shown in Figure 5.11. The algorithm is based on QGMM scheduling approach proposed by He et al. [60]. Initially, a set of potential resources meeting the availability and bandwidth requirements of each application task is identified. A task–resource match providing minimum MCT (Min-Min) among other task–resource pairs is scheduled first. After assignment of tasks (with QoS requests), the scheduler maps the remaining tasks (without QoS requests) over the grid resources while minimizing the earliest completion time. A real scenario of simulated environment for scheduling application tasks is illustrated in Figures

5.8 to 5.10. The simulation parameters for an application and Grid computing system are shown in Table 5.2.

Table 5.2: Simulation parameters for schedules shown in Fig. 5.8 to 5.10

Number of grid resources	[4]
Number of independent tasks	[8]
Resource availability	[0.25 0.55 0.75 1.0]
Resource bandwidth availability	[300 500 600 1000]
Task availability demand	[0.9 0.5 0.75 0.8 0.5 0.95 0.8 0.9]
Task bandwidth demand	[100 800 500 400 350 675 700 250]
%age of dedicated resources in grid	[25%]

Table 5.3: Computation cost matrix for schedules shown in Fig. 5.8 to 5.10

	P1	P2	P3	P4
Task1	50	80	40	27
Task2	25	40	20	14
Task3	13	20	10	7
Task4	25	40	20	14
Task5	50	80	40	27
Task6	25	40	20	14
Task7	25	40	20	14
Task8	13	20	10	7

In this example, a grid system consists of four resources where 25% resources are dedicated. An application consisting of 8 tasks has been considered for scheduling. Each task has QoS demands for availability and bandwidth. Figure 5.8 presents a schedule using AQuA approach where makespan is 76 and availability surplus is 14.4% while meeting 100% QoS requests of tasks. In Figure 5.9, a schedule is generated with QGMM heuristic which ignores the availability concerns of tasks. It yields a schedule of same makespan (i.e., makespan=76) but meets only 62% QoS requirements. The system availability surplus is 5.6% which is very less as compared to AQuA. Table 5.3 shows task computation time on different processors for application and grid modeling parameters given in Table 5.2.

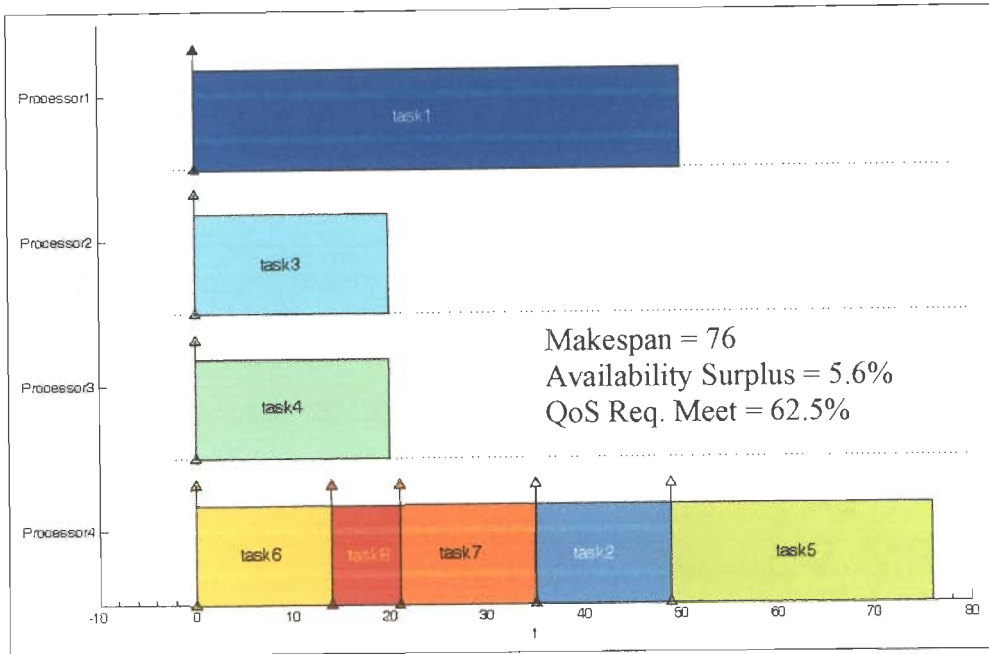


Figure 5.9: Gantt chart showing schedule of QGMM

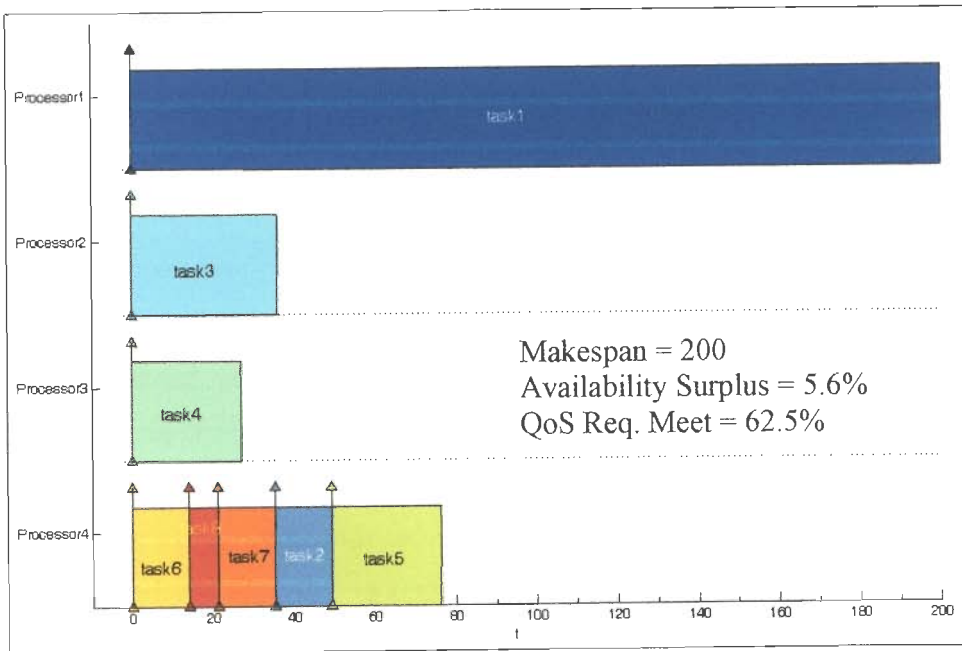


Figure 5.10: Gantt chart showing schedule of QGMM-A

In Figure 5.10, schedule is generated using a variation of QGMM known as QGMM–A which includes availability overhead of resources. In QGMM–A, initial schedule is obtained using QGMM and then, schedule length on each resource is modified using Equation (5.16) to add the effect of resource availability. In this case, the schedule length (makespan) increases to 200 which prove that QGMM is ineffective where resources exhibit different availability patterns.

Algorithm 5.4: AQuA

All tasks n_i in T are grouped into sets T_1 and T_2 . Set T_1 consists of m_1 tasks with QoS requests while set T_2 consists of m_2 tasks without QoS requests.

1. **for** all tasks in set T (in an arbitrary order)
2. **for** all resources (in a fixed arbitrary order)
3. $CT_{ij} = AT(r_j) + ETC_{ij}$
4. **end for**
5. **end for**
6. **do until** there is any task in T_1
7. **for** each task n_i
8. **find** a resource r_j in the QoS qualified resource set that obtains the earliest completion time
9. **end for**
10. **find** a task n_i with minimum earliest completion time (ECT)
11. **assign** task n_i to the resource r_j that gives minimum ECT
12. **delete** task n_i from T_1
13. **update** the available time $AT(r_j)$ of resource r_j
14. **update** CT_{ij} for task n_i
15. **end do**
16. **do until** there is any task in T_2
17. **for** each task n_i
18. **find** a resource r_j in the resource set that obtains the earliest completion time
19. **end for**
20. **find** a task n_i with minimum earliest completion time (ECT)
21. **assign** task n_i to the resource r_j that gives minimum ECT
22. **delete** task n_i from T_2
23. **update** the available time of resource r_j
24. **update** CT_{ij} for task n_i
25. **end do**

Figure 5.11: The pseudo code of AQuA algorithm

Considering availability as QoS eliminates the risks of tasks to be scheduled on unreliable resources and encourages highly available resources to participate in scheduling to provide robustness, stability and reliability of grid computing systems.

5.3.6 Simulation and Result Analysis

The proposed algorithm (AQuA) has been validated in the simulated grid environment and the results have been compared with the existing QGMM [60] heuristic and QGMM–A approach (a variation of QGMM including availability overhead). The experimental results have been depicted in Figures 5.12 – 5.19 that illustrate the comparative analysis for the performance metrics i.e. availability surplus of resources and makespan. Different scenarios have been used in simulation testing which affect the performance of task scheduling. Table 5.4 shows the parameters of simulated grid computing systems.

In what follows, we briefly introduce the performance metrics used to evaluate the performance of proposed availability–aware scheduling strategy:

1. *Availability Surplus*. It is the excess amount of availability of a resource as computed to the availability requested from a task scheduled over it. Availability surplus of a system quantifies the excess of availability offered over demand.
2. *Makespan*. It is duration of time between the start time of first task and the finish time of last task in a schedule.
3. *Percentage of tasks meeting QoS*. It is a ratio of tasks meeting QoS requests to the total number of tasks.

Table 5.4: Simulation setup parameters

Number of grid resources	[100, 500]
Number of independent tasks	[200, 1000]
Resource availability	[0.1, 1.0]
Resource bandwidth availability	[10, 1000] Mbps
Task availability demand	[0.1, 1.0]
Task bandwidth demand	[10, 1000] Mbps
%age of dedicated resources in grid	[5%, 50%]
Task Computation Time	[1, 100]
Resource Capacity	[0.1, 1.0]

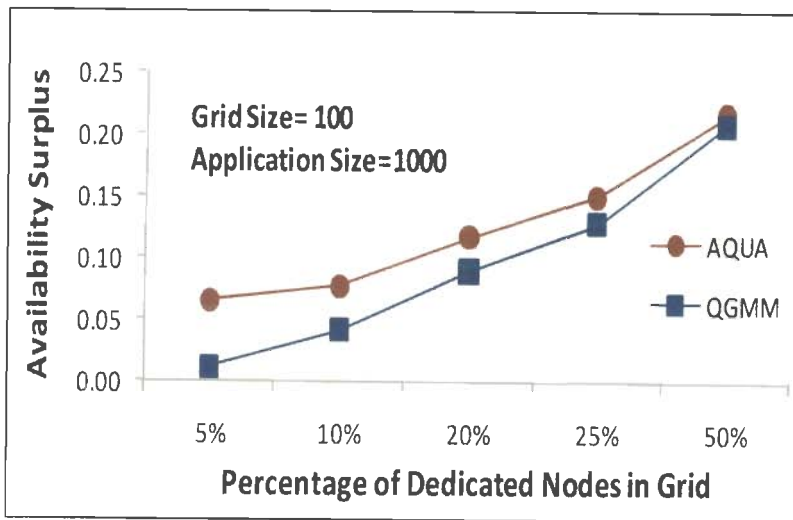


Figure 5.12: Performance in terms of availability surplus w.r.t. dedicated nodes

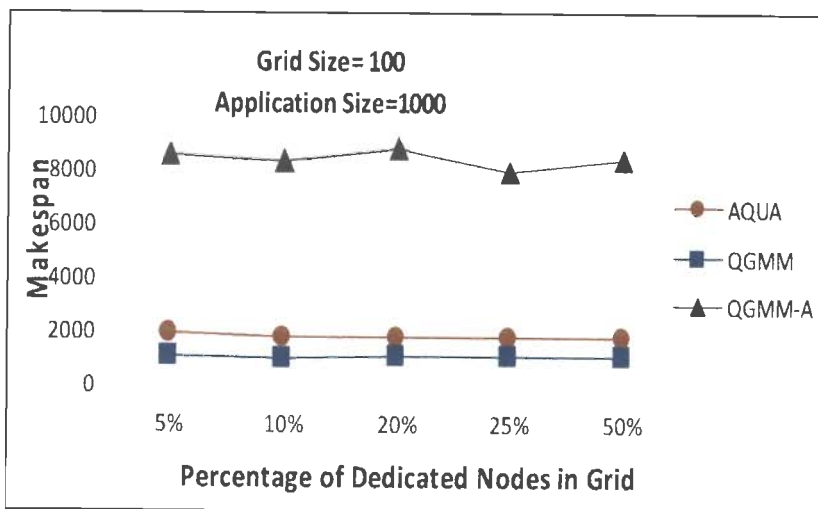


Figure 5.13: Performance in terms of makespan w.r.t. dedicated nodes

In first group of experiments, we vary the percentage of dedicated nodes from 5% to 50%. In Figure 5.12, the comparative results show that AQUA produces better availability surplus than QGMM when non-dedicated resources are larger in a grid system (Tasks = 1000, Resources = 100) which is true for dynamic grids where non-dedicated resources are usually large. A node with 100% availability (availability level=1) connected over a network of bandwidth no less than 1 Giga bits per second is considered as dedicated node. In Figure 5.13, the makespan is compared for AQUA, QGMM and QGMM-A.

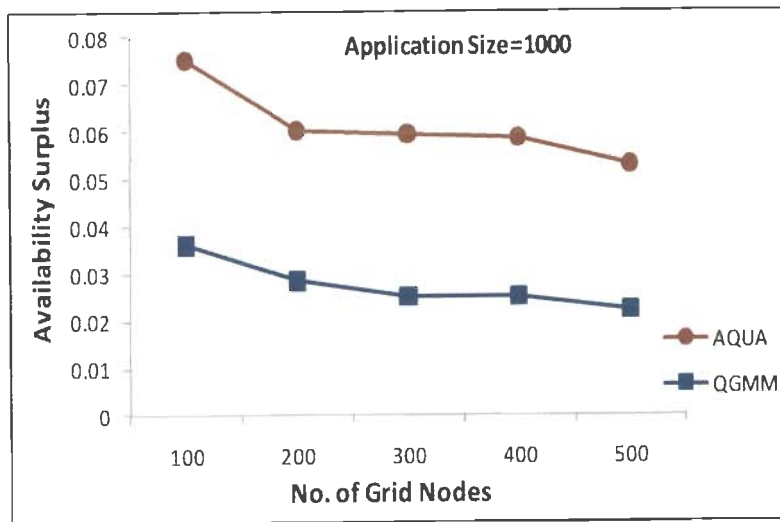


Figure 5.14: Performance in terms of availability surplus w.r.t. grid size

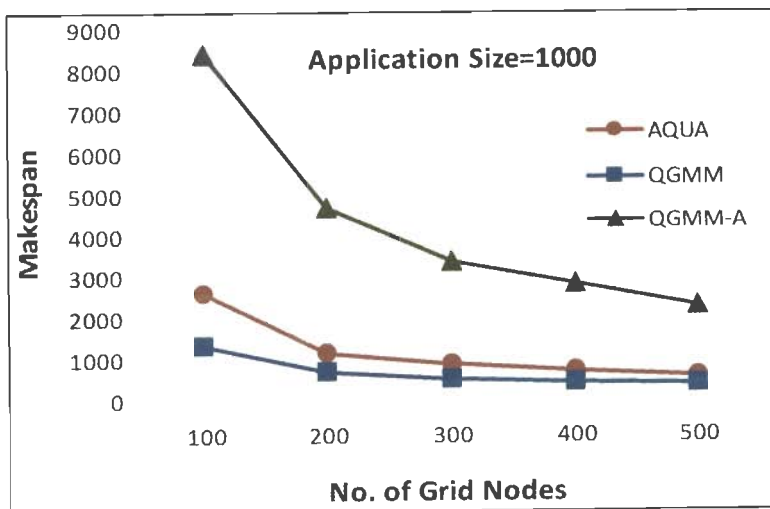


Figure 5.15: Performance in terms of makespan w.r.t. grid size

In general, QGMM is able to produce better makespan than AQuA as it is not considering availability concerns. AQuA generates poor schedules due increased ETC (Equation (5.17)) whereas QGMM-A generates schedules which are poor as compared to AQuA. Therefore, AQuA is able to obtain highly available resources in scheduling without affecting the makespan too adversely.

The second group of experiments is focused on the scalability of grid system. The same application of 1000 tasks is tested to be scheduled on a grid system with varying number of nodes i.e., 100 to 500. In Figure 5.14, the results show that availability surplus is high initially when grid size is small and it is decreasing with the increase of grid size. AQuA has increased availability surplus than QGMM. As grid size increases, a set of potential resources increases for each task so that each task has better option to select a resource which can minimize the task completion time. The Figure 5.15 justifies the above statement. It shows that QGMM has better makespan than AQuA which decreases with the increase in number of grid resources, but QGMM-A generates unpredictable schedules of poorer makespan than AQuA. Another comparison is shown in Figure 5.16 which indicates that AQuA meets 100% QoS whereas QGMM meets nearly 50% QoS requests.

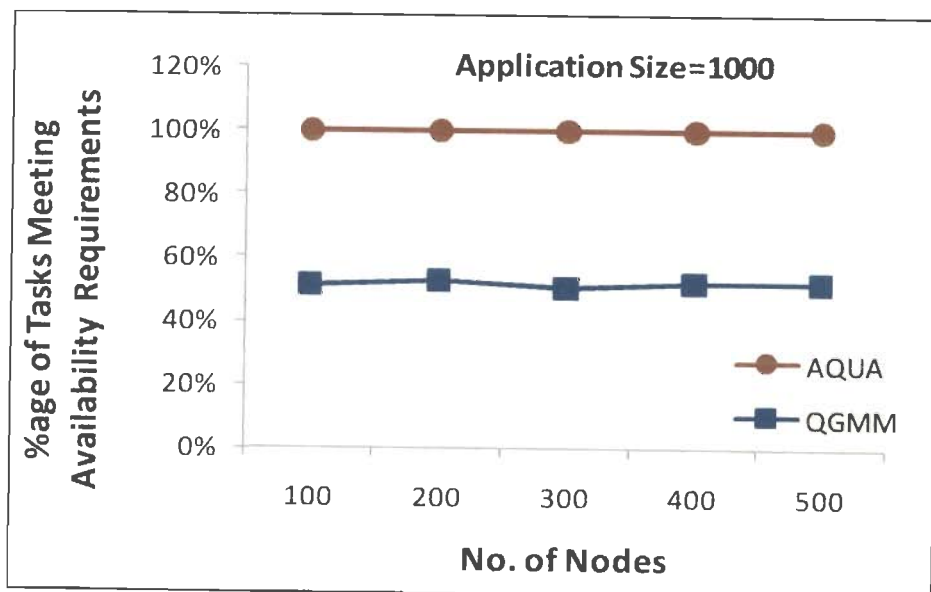


Figure 5.16: Performance in terms of QoS satisfaction w.r.t. grid size

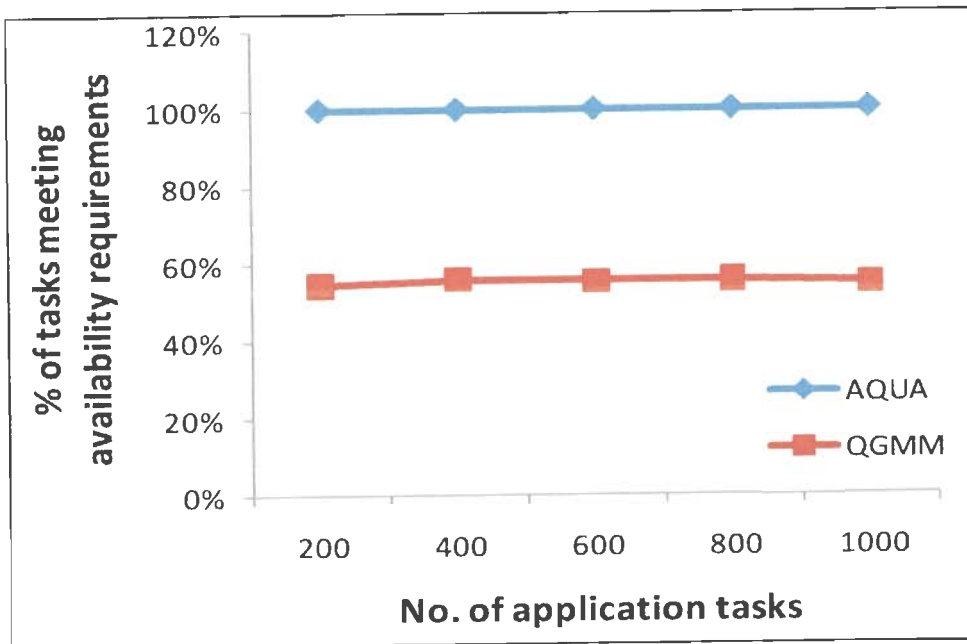


Figure 5.17: Performance in terms of QoS satisfaction w.r.t. application size

The next group of experiments is modeled on the scalability of application size. The Figure 5.17 indicates that AQUA meets the QoS of all tasks while QGMM has approximately 50% QoS qualified tasks. Figure 5.18 shows AQUA yields higher availability surplus than QGMM which tends to decrease as application size increases. The results shown in Figure 5.19 indicate that makespan increases with the increase in application sizes (grid size is fixed at 100 nodes). The result analysis shows that QGMM produces better schedules than AQUA but its variation QGMM-A generates very poor schedule than AQUA. The results show that AQUA produces better availability as number of tasks with QoS requirements increases without affecting the makespan. Thus, the AQUA algorithm generates more reliable schedule than QGMM in highly dynamic grid environment at small cost of makespan which is affordable to lessen the risks associated for executing the applications in grids.

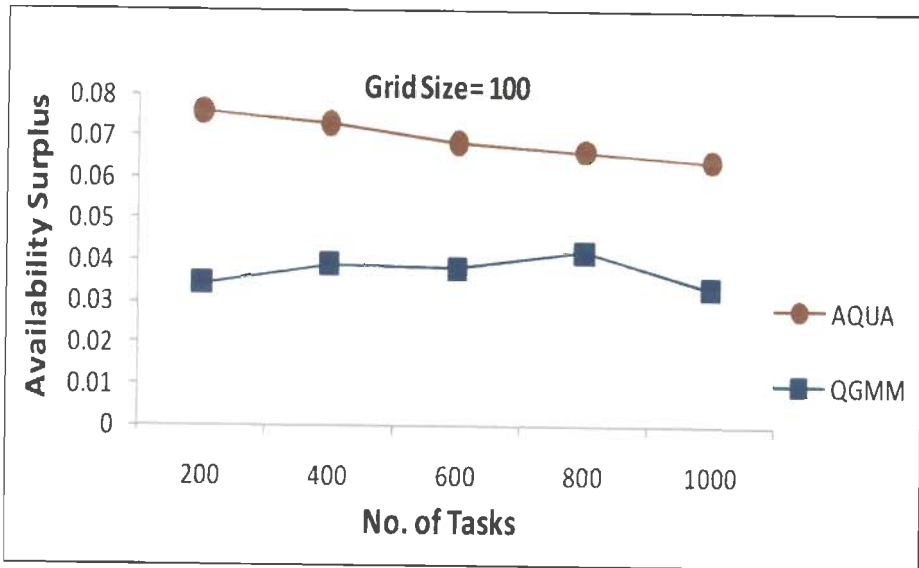


Figure 5.18: Performance in terms of availability surplus w.r.t. application size

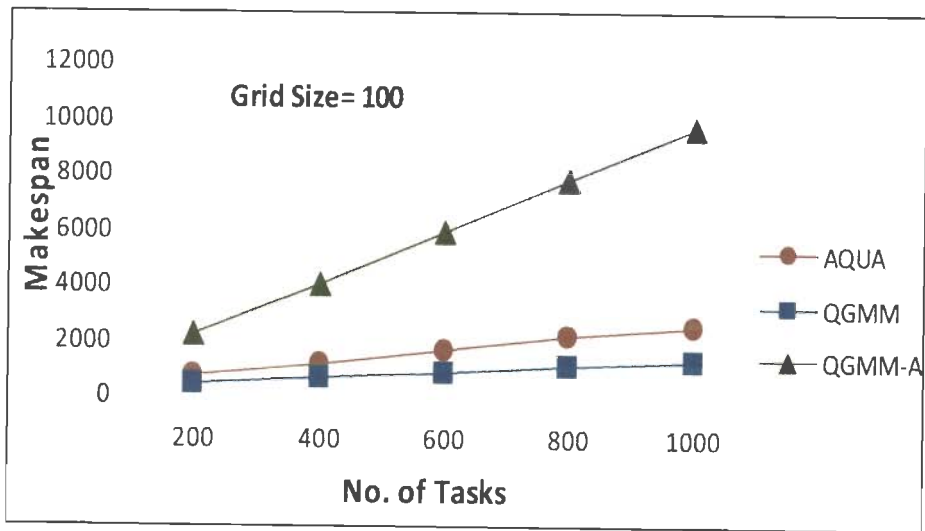


Figure 5.19: Performance in terms of makespan w.r.t. application size

5.4 Summary and Discussions

New challenges have been raised with the evolution of grid computing systems. The objectives need to be revised. To obtain a schedule with

minimum makespan is still an objective but the nature of resources in grid is highly dynamic and unreliable. This research work addresses the notion of trust in selecting hopefully trustworthy resources and executing tasks on these resources in Grid computing environment. The trust criterion has been combined with other criterion like cost and execution time for considering such a multi criteria scheme for Grid scheduling algorithms. We have presented a novel scheduling algorithm called Trust-oriented Multi-objective Scheduling algorithm (Trust-MOS) which optimizes both the makespan and economic cost of the schedule while maximizing the total trust (or reliability) of Grid resources for executing applications. Our approach remarkably lessens the risks in task scheduling while considering multiple criteria and reduces task failure ratio that can be best suited for executing tasks in a secure manner in Grids.

Similarly, selection of highly available resources is an important issue while scheduling the tasks. We have proposed an effective scheduling algorithm (AQuA) which discovers the resources meeting the QoS requirements of tasks and then schedules the task over the resources minimizing the overall completion time (makespan) while maximizing the availability of grid systems. We have addressed two-dimensional QoS issues i.e., availability and bandwidth requirements of tasks. These constraints have major impact on the performance of scheduling applications in grid computing systems. The QoS components are added to the traditional Min-Min heuristic to form the proposed heuristic (AQuA). A simulated grid scheduling system is developed to test the performances of AQuA and QGMM. The result analysis shows that AQuA algorithm gives preference to highly available resources in scheduling tasks while meeting the QoS requirements and increases the availability of grid to successfully execute applications without adversely affecting the makespan.

6.1 Conclusions

In this research, we have presented both single criterion and multiple criteria scheduling algorithms for executing workflows (one workflow after another) in grids. We have proposed duplication based single criterion economic scheduling algorithms for homogeneous (RD) and heterogeneous (HED) environments. These algorithms adopt an efficient duplication strategy to optimize makespan and to reduce average processor consumption after minimizing the number of duplications and removal of unproductive schedules over different resources. These algorithms prove their usefulness over existing duplication based scheduling heuristics for heterogeneous (e.g., Grids) and homogeneous (e.g., Clusters) computing systems. Another single criteria scheduling algorithm i.e., SHCP has been proposed for grid computing systems. In this, a heterogeneity model has been suggested to account for the heterogeneity of computational resources and communication networks in grid. A critical path based task sequence is generated considering the processor and network heterogeneity factors for workflow applications. This strategy is able to

generate a better task sequence for producing shorter schedules for large workflows.

Coming to multi criteria, two stage bi-criteria scheduling approaches (DBSA, SODA) are suggested to fairly optimize both the economic cost and makespan for executing DAG applications (one DAG after another) in the grid environments. At primary stage, duplication strategy has been exploited in an intelligent way to find better schedules in terms of execution time. In secondary stage, the above schedules are investigated to minimize the economic cost without affecting the makespan or sliding (or increasing) the makespan up to a specified limit. The comparative analysis shows the goodness of these heuristics. The duplication strategy has been rarely used in bi-criteria scheduling which has been used intelligently to optimize both economic cost and execution time.

Trust is also equally important factor for users and resources while allocating the tasks on to the resources in the grid. If the resources are not trustworthy then the results may be suspicious and applications may be damaged. Therefore, resources must be selectively chosen which exhibit high trust levels. A trust-oriented multiple criteria scheduling algorithm (Trust-MOS) for executing workflow tasks in a secure way over grid resources has been proposed. The results show the schedules obtained using Trust-MOS are more reliable as they are schedules over trustworthy resources at the small cost of execution time to the users.

Further, the research has been focused towards considering availability and bandwidth of resources as QoS constraints (AQuA) in scheduling. Extensive simulations of above scheduling algorithms indicate their fitness in the current grid scheduling systems. The comparative result analysis reveals that these scheduling strategies can be better utilized to increase the performance of grid scheduling system and also able to reduce economic cost for grid users while providing the potential resources for running applications.

6.2 Future Research Directions

The research work can be extended to analytically investigate the proposed algorithms in the dynamic heterogeneous environment where processor load, capability and network condition vary during the execution of workflow applications. The workflow paradigm has become the standard to represent the scientific applications and their execution flows. The need of a scheduler which deals with multiple workflows is an open problem in grids. Optimizing multiple objectives in scheduling over multiple workflows makes this problem even harder. There exist very few scheduling approaches which can optimize the execution of more than one workflow at a time [144]. A future research work can be to investigate new efficient multi-criteria scheduling strategies for executing multiple workflows in grid computing environments.

The multi criteria scheduling technique for executing multiple scientific workflows in grids should have following objectives:

- The scheduling algorithm must be able to schedule tasks in multiple workflows based on resources required to avoid bottlenecks and delays.
- The constraints (e.g., deadline and budget) of every workflow should be satisfied.
- Application objectives (execution time, economic cost) should be minimized while maximizing the trust and reliability of executing application on the grids.

BIBLIOGRAPHY

- [1] Abawajy, J. H., "Fault-Tolerant Scheduling Policy for Grid Computing Systems", *Proc. of IEEE Int'l. Parallel and Distributed Symposium*, April 2004.
- [2] Abdul-Rahman, A., and Hailes, S., "Supporting Trust in Virtual Communities", *Proc. of the 33rd Hawaii Int'l Conference on System Sciences*, January 2000.
- [3] Abraham, A., Buyya, R., and Nath, B., "Nature's Heuristics for Scheduling Jobs on Computational Grids", *Proc. of 8th IEEE Int'l Conference on Advanced Computing and Communications (ADCOM 2000)*, pp. 45–52, Cochin, India, December 2000.
- [4] Adar, E., and Huberman, B., "Free Riding on Gnutella", *First Monday*, 5(10), 2000.
- [5] Ahmed, I., and Kwok, Y. -K., "On Exploiting Task Duplication in Parallel Program Scheduling", *IEEE Transactions on Parallel and Distributed Systems*, 9(9): pp. 872–892, Sept 1998.
- [6] Azzedin, F., and Maheswaran, M., "Integrating Trust into Grid Resource Management Systems", *Proc. of the 2002 Int'l Conference on Parallel Processing (ICPP 2002)*, pp. 47–54. IEEE Press, Canada, 2002.
- [7] Bajaj, R., and Agrawal, D. P., "Improving Scheduling of Tasks in A Heterogeneous Environment", *IEEE Transactions on Parallel and Distributed Systems*, vol.15, no. 2, pp.107–118, February 2004.
- [8] Bansal, S., Kumar, P., and Singh, K., "An improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor

- Systems”, *IEEE Transactions on Parallel and Distributed Systems*, 14(6), pp. 533–544, 2003.
- [9] Bansal, S., Kumar, P., and Singh, K., “Dealing with Heterogeneity through Limited Duplication for Scheduling Precedence Constrained Task Graphs”, *Journal of Parallel and Distributed Computing*, 65(4), pp. 479–491, April 2005.
- [10] Baraglia, R., Ferrini, R., Ricci, L., Tonello, N., and Yahyapour, R., “QoS-constrained List Scheduling Heuristics for Parallel Applications on Grids”, Julien Bourgeois and Didier El Baz editors, *Proc. of the 16th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP2008)*, IEEE Computer Society Press, Toulouse, France, February 2008.
- [11] Baruah, S., and Burns, A., “Sustainable Scheduling Analysis”, *Proc. of the 27th IEEE Real-Time Systems Symposium*, pp. 159–168, 2006.
- [12] Benjamin, K. B. T., and Bharadwaj, V., “An Adaptive Co-ordinate based Scheduling Mechanism for Grid Resource Management with Resource Availabilities for Grid Computing Environments”, *To appear in an edited Book by Fatos Xhafa and Ajith Abraham on Meta-heuristics for Scheduling in Emergent Computational Systems*, Series in Studies in Computational Intelligence, Springer-Verlag, USA, 2008.
- [13] Benjamin, K. B. T., Bharadwaj, V., Terence Hung, and Simon See, “A Multi-Dimensional Scheduling Scheme in a Grid Computing Environment”, *Journal of Parallel and Distributed Computing (JPDC)*, vol. 67, no. 6, pp. 659–673, 2007.
- [14] Bernat, G., and Burns, A., “Three Obstacles to Flexible Scheduling”, *Proc. of the 13th Euromicro Int’l Conference on Real Time Systems*, pp. 11–18, 2001.

- [15] Berryman, K., Herbermann, J., and Richardson, J., "An Internet based System to Reduce Travel Costs, Enhance Human Resource Services, and reduce overall IT cost", *A Report on Application Service Providers*, pp. 1 – 16, September 28, 2006.
- [16] Bertino, E., Crispo, B., and Mazzoleni, P., "Supporting Multi-Dimensional Trustworthiness for Grid Workflows", *DELOS Workshop: Digital Library Architectures*, pp. 195–204, 2004.
- [17] Bote-Lorenzo, M. L., Dimitriadis, Y. A., and Gomez-Sanchez, Eduardo, "Grid Characteristics and Uses: a Grid Definition", *First European Across Grids Conference*, pp. 291–298, 2004.
- [18] Bozdag, D., Ozguner, F., and Catalyurek, U., "Compaction of Schedules and a Two-stage Approach for Duplication-based DAG Scheduling", *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857–871, 2009.
- [19] Braun, R., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., Yao, B., Hensgen, D., and Freund, R., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems", *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.
- [20] Braun, T. D., Siegel, H. J., and Beck et al., "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems", *IEEE Workshop on Advances in Parallel and Distributed Systems*, West Lafayette, pp. 330–335, 1998.
- [21] BURP, <http://burp.renderfarming.net/>

- [22] Butt, A. R., Adabala, S., Kapadia, N. H., Figueiredo, R. J., and Fortes, J. A. B., "Grid-Computing Portals and Security Issues", *Journal of Parallel and Distributed Computing*, 63(10), 1006–1014, 2003.
- [23] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F., "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments", *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, pp. 349–363, Cancun, Mexico, May 2000.
- [24] Casanova, H., Obertelii, G., Berman, F., and Wolski, R., "The AppLeS Parameter Sweep Template: User-level Middleware for the Grid", *Proc. the Super Computing Conference (SC'2000)*, 2000.
- [25] Casavant, T., and Kuhl, J., "A Taxonomy of Scheduling in general Purpose Distributed Computing Systems", *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, 1988.
- [26] Chong, A., Sourin, A., and Levinski, K., "Grid-based Computer Animation Rendering", *Proc. of the 4th Int'l Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, Malaysia, GRAPHITE '06, ACM, New York, 39–47, 2006.
- [27] Chung, Y. -C., and Ranka, S., "Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors", *Proc. on Supercomputing*, pp. 512–521, Nov 1992.
- [28] Climateprediction.net, <http://climateprediction.net/>
- [29] Daoud, M. I., and Kharma, N. N., "Efficient Compile-Time Task Scheduling for Heterogeneous Distributed Computing Systems", *Proc. of the 12th IEEE Int'l Conference on Parallel and Distributed Systems*, pp. 11–22, 2006.

- [30] Daoud, M., and Kharma, N., "GATS 1.0: a novel GA-based scheduling algorithm for task scheduling on heterogeneous processor nets", *Proc. of the 2005 Conference on Genetic and Evolutionary Computation (GECCO '05)*, Hans-Georg Beyer (Ed.), ACM, New York, NY, USA, 2209–2210, 2005.
- [31] DaSilva, L., and Srivastava, V., "Node Participation in Ad Hoc and Peer-to-Peer Networks: A Game-Theoretic Formulation", *Workshop on Games and Emergent Behavior in Distributed Computing Environments*, Birmingham, 2004.
- [32] De Falco, I., Del Balio, R., Tarantino, E. and Vaccaro, R., "Improving Search by Incorporating Evolution Principles in Parallel Tabu Search", *IEEE Conference on Evolutionary Computation*, 1994.
- [33] Deelman, E., Blythe, J., Gil, Y., and Kesselman, C., "Workflow Management in GriPhyN", *Grid Resource Management, State of the Art and Future Trends*, pp. 99–116, 2004.
- [34] DiMasi, J. A., "The Price of Innovation: New Estimates of Drug Development Costs", *Journal of Health Economics*, (22):151–185, 2003.
- [35] Doboli, A. and Eles, P., "Scheduling Under Data and Control Dependencies for Heterogeneous Architectures", *Proc. of the Int'l Conference on Computer Design (ICCD'98)*, pp. 602–608, Austin, Texas USA, October 1998.
- [36] Dogan, A. and Ozguner, F., "Biobjective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems", *Journal of Computers*, vol. 48, no. 3, pp. 300–314, 2005.

- [37] Dogan, A. and Ozguner, F., "LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems", *Proc. of the Int'l Conference on Parallel Processing (ICPP'02)*, pp. 352–359, Aug 2002.
- [38] Dogan, A. and Ozguner, F., "Scheduling Independent Tasks with QoS requirements in Grid Computing with Time-varying Resource Prices", *Proc. of GRID 2002*, Lecture Notes in Computer Science, vol. 2536, Springer, Berlin, pp. 58–69, 2002.
- [39] DrugDiscovery@Home, <http://www.drugdiscoveryathome.com/>
- [40] Einstein@Home, <http://www.einsteinathome.org/>
- [41] El-Rewini, H., Lewis, T., and Ali, H., "Task Scheduling in Parallel and Distributed Systems", ISBN: 0130992356, PTR Prentice Hall, 1994.
- [42] Ernemann, C., Hamscher, V., and Yahyapour, R., "Economic Scheduling in Grid Computing", *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, LNCS, vol. 2537, Springer, pp. 128–152, 2002.
- [43] Folding@Home, <http://folding.stanford.edu/>
- [44] Foster, I., and Kesselman, C., "The GRID 2: Blueprint for a New Computing Infrastructure", *Morgan Kaufmann Publishers*, Elsevier Inc., 2004.
- [45] Foster, I., Fidler, M., Roy, A., Sander, V., and Winkler, L., "End-to-end Quality of Service for High-end Applications", *Elsevier Comput. Comm. J.*, 27(14), pp. 1375–1388, 2004.
- [46] Foster, I., Roy, A., and Sander, V., "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation", *Proc. of*

the 8th Int'l Workshop on Quality of Service, Pittsburgh, PA, USA, pp. 181–188, June 5–7, 2000.

- [47] Franke, C., Lepping, J., and Schwiegelshohn, U., "Greedy Scheduling with Complex Objectives", *Proc. of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, pp. 113–120, IEEE Press, April, 2007.
- [48] Fusion Grid, <http://www.fusiongrid.org/>
- [49] Gao, Z., Luo, S., and Ding, D., "A Scheduling Mechanism Considering Simultaneously Running of Grid Tasks and Local Tasks in the Computational Grid", *Proc. of Int'l Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, 2007.
- [50] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., "PVM: Parallel Virtual Machine", *MIT press*, 1994.
- [51] Gerasoulis, A., and Yang, T., "A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors", *Journal of Parallel and Distributed Computing*, 16(4): pp. 276–291, 1992.
- [52] Golconda, K. S., and Ozguner, F., "A Comparison of Static QoS-based Scheduling Heuristics for a Meta-task with Multiple QoS Dimensions in Heterogeneous Computing", *Proc. of the 18th Int. Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.
- [53] Gong, L., Sun, X. H., and Waston, E., "Performance Modeling and Prediction of Non-dedicated Network Computing", *IEEE Transactions on Computer*, 51(9): 1041–1055, 2002.
- [54] Grimme, C., Lepping, J., and Papaspyrou, A., "Discovering Performance Bounds for Grid Scheduling by using Evolutionary Multiobjective Optimization", *Proc. of the Genetic and Evolutionary Computation*

Conference (GECCO 2008), pp. 1491–1498, Atlanta, Georgia, USA, ACM Press, July, 2008.

- [55] Gupta, A., Ahire, S., Greenwood, G., and Terwilliger, M., “Workforce–Constrained Preventive Maintenance Scheduling using Evolution Strategies”, *Decision Sciences*, 31(4):1–27, 2001.
- [56] Gupta, A., Greenwood, G., and McSweeney, K., “Scheduling Tasks in Multiprocessor Systems using Evolutionary Strategies”, *Proc. of the IEEE Int’l Conference on Evolutionary Computation*, pp. 345–349, 1994.
- [57] Gupta, A., Greenwood, G., and Terwilliger, M., “Scheduling Replicated Critical Tasks in Faulty Networks Using Evolutionary Strategies”, *Proc. of the 1995 IEEE Int’l Conf. on Evolutionary Computing*, pp. 152–156, 1995.
- [58] Hagrais, T., and Janecek, J., “A High Performance Low Complexity Algorithm for Compile–Time Job Scheduling in Homogeneous Computing Environments”, *Proc. of Int’l Conference on Parallel Processing Workshops*, pp. 149–155, Oct 2003.
- [59] He, L., Jarvis, S. A., Spooner, D. P., Bacigalupo, D., Tan, G., and Nudd, G. R., “Mapping DAG–based Applications to Multiclusters with Background Workload”, *Proc. of IEEE Int’l Symposium on Cluster Computing and the Grid (CCGrid’05)*, pp. 855–862, May 2005.
- [60] He, X., Sun, X., and Laszewski, G., “QoS Guided Min–Min Heuristic for Grid Task Scheduling”, *Journal of Computer Science and Technology, Special Issue on Grid Computing*, vol. 18, no. 4, pp. 442–451, July 2003.
- [61] Heap, D., “Scorpion: Simplifying the Corporate IT Infrastructure”, *IBM® Research White Paper*, 2000.

- [62] Hou, E., Ansari, N., and Ren, H., "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, Feb 1994.
- [63] Iverson, M., and Ozguner, F., "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment", *Proc. of Seventh Heterogeneous Computing Workshop*, pp. 70–78, Orlando, Florida USA, March 1998.
- [64] Kamvar, S. D., Schlosser, M. T., and Garcia–Molina, H., "The Eigentrust Algorithm for Reputation Management in P2P Networks", *Proc. of the 12th Int'l Conference on World Wide Web*, pp. 640–651, 2003.
- [65] Khan, A. A., McCreary, C., and Jones, M. S., "A Comparison of Multiprocessor Scheduling Heuristics", *ICPP*, pp. 243–250, 1994.
- [66] Khokhar, A. A., Prasanna, V. K., Shaaban, M. E., Wang, C.-L., "Heterogeneous Computing: Challenges and Opportunities", *IEEE Comput.*, 26 (6), 18–27, 1993.
- [67] Kim, S. C., Lee, S., and Hahm, J., "Push–Pull: Deterministic Search–Based DAG Scheduling for Heterogeneous Cluster Systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 11, November 2007.
- [68] Kirkpatrick, S., Gelatt, C. D. Jr., and Vecchi, M. P., "Optimization using Simulated Annealing", *Science Journal*, vol. 220, no. 4598, 1983.
- [69] Kruatrachue, B., and Lewis, T.G., "Grain Size Determination for Parallel Processing", *IEEE Software*, 5(1), pp. 23–32, Jan 1988.
- [70] Kung Fu Panda, <http://www.kungfupanda.com/>

- [71] Kuruwski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A., and Pukaeki, J., "Improving Grid Level Throughput Using Job Migration And Rescheduling", *Scientific Programming*, vol. 12, no. 4, pp. 263–273, 2004.
- [72] Kwok, Y., and Ahmed, I., "Benchmarking and Comparison of the Task Graph Scheduling Algorithms", *Journal of Parallel and Distributed Computing*, 59(3): 381–422, 1999.
- [73] Lai, Kuan–Chou, and Yang, Chao–Tung, "A Dominant Predecessor Duplication Scheduling Algorithm for Heterogeneous Systems", *Journal of Supercomputing*, 44(2): pp. 126–145, 2008.
- [74] Lanfermann, G., Allen, G., Radke, T., and Seidel, E., "Nomadic Migration: A New Tool for Dynamic Grid Computing", *Proc. of the 10th IEEE Int'l Symposium on High Performance Distributed Computing (HPDC'01)*, pp. 429–430, San Francisco, California USA, August 2001.
- [75] Lee, C.–Y., "Two–Machine Flowshop Scheduling with Availability Constraints", *European J. Operational Research*, vol. 114, no. 2, pp. 420–429, April 1999.
- [76] Lee, L.–T., Chang, H. –Y., Liu, K. –Y., Chang, G. –M., and Lien, C. –C., "A Dynamic Scheduling Algorithm in Heterogeneous Computing Environments", *Int'l Sym. on Communications and Information Technologies (ISCIT'06)*, pp. 313–318, 2006.
- [77] Li, Chunlin, and Li, Layuan, "A Pricing Approach for Grid Resource Scheduling with QoS Guarantees", *Fundamenta Informaticae*, 76(2007), pp. 59–73, 2007.

- [78] Li, Chunlin, and Li, Layuan, "Utility-based QoS Optimization Strategy for Multi-criteria Scheduling in Grid", *Journal of Parallel and Distributed Computing*, 2006.
- [79] Li, H., and Singhal, M., "Trust Management in Distributed Systems", *Computer*, vol. 40, no. 2, pp. 45–53, 2007.
- [80] Lin, Wei-Ming, and Gu, Q., "An Efficient Clustering-Based Task Scheduling Algorithm for Parallel Programs with Task Duplication", *Journal of Information Science and Engineering*, vol. 23, no. 2, pp. 589–604, 2007.
- [81] Liou, J., and Palis, M. A., "A Comparison of General Approaches to Multiprocessor Scheduling", *Proc. of the 11th Int'l Symposium on Parallel Processing*, pp. 152–156, April 1997.
- [82] Liou, J., and Palis, M. A., "An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors", *Proc. of Workshop on Resource Management, Symposium of Parallel and Distributed Processing*, pp. 152–156, Oct 1996.
- [83] Luo, Y., Xue, Y., and Zhong, S., "Road Extraction from IKONOS Image using Grid Computing Platform", *Proc. of IEEE Int'l Geoscience and Remote Sensing Symposium (IGARSS '05)*, vol. 6, no., pp. 3895– 3898, July 2005.
- [84] Ma, T., and Buyya, R., "Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids", *Proc. of the 17th Int'l Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, Brazil, October 2005*.
- [85] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., and Freund, R. F., "Dynamic Matching and Scheduling of a Class of Independent Tasks

onto Heterogeneous Computing Systems”, *Journal of Parallel and Distributed Computing*, vol. 59, No. 2, pp. 107–131, November 1999.

- [86] MilkyWay@Home, <http://milkyway.cs.rpi.edu/milkyway/>
- [87] MPI Forum, “MPI: A Message Passing Interface Standard”, *International Journal of Supercomputer Application*, 8 (3/4), pp. 165–416, 1994.
- [88] Muthuvelu, N., Liu, J., Soe, N. L., Venugopal, S., Sulistio, A., and Buyya, R., “A Dynamic Job Grouping-based Scheduling for Developing Applications with Fine-Grained Tasks on Global Grids”, *Proc. of the Third Australasian Workshop on Grid Computing and e-Research (AusGrid 2005)*, Newcastle, Australia, 2005.
- [89] Nadeem, F., Prodan, R., and Fahringer, T., “Characterizing, Modeling and Predicting Dynamic Resource Availability in a Large Scale Multi-Purpose Grid”, *Proc. of CCGrid*, Lyon, France, 2008.
- [90] Olivier, B., Vinecent, B., and Yves, R., “The Iso-level Scheduling Heuristic for Heterogeneous Processors”, *Proc. of the 10th Euromicro workshop on parallel, distributed and network-based processing*, pp. 335–342, 2002.
- [91] Papadimitriou, C. H., and Yannakakis, M., “Towards an Architecture-Independent Analysis of Parallel Algorithms”, *SIAM Journal of Computing*, 19(2), pp. 322–328, April 1990.
- [92] Park, G.-L., Shirazi, B., and Marquis, J., “DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems”, *Proc. of the 11th Int'l Parallel Processing Symposium*, pp. 157–166, April 1997.
- [93] Pasham, S., and Lin, Wei-Ming, “Efficient Task Scheduling with Duplication for Bounded Number of Processors”, *The IEEE 11th Int'l*

Conference on Parallel and Distributed Systems (ICPADS–2005), Fukuoka, Japan, 2005.

- [94] Pasham, S., and Lin, Wei–Ming, “Task Scheduling Algorithm with Duplication for Distributed Computing”, *17th Int’l Conference on Computer Applications in Industry and Engineering*, Orlando, FL., 2004.
- [95] Pinedo, M., “Scheduling: Theory, Algorithms and Systems”, *Prentice Hall*, 1995.
- [96] Qin, X., and Xie, T., “An Availability–Aware Task Scheduling Strategy for Heterogeneous Systems”, *IEEE Transaction on Computers*, vol. 57(2), pp. 188–199, 2008.
- [97] Quinn, M. J., “Parallel Computing: Theory and Practice”, *Tata McGRAW Hill*, 1994.
- [98] Radulescu, A., and Gemund, A. J. C. van, “On the Complexity of List Scheduling Algorithms for Distributed Memory Systems”, *Proc. of 13th Int’l Conference on Supercomputing*, pp. 68–75, Portland, Oregon, USA, November 1999.
- [99] Radulescu, A., Gemund, A.J.C. van, “Fast and Effective Task Scheduling in Heterogeneous Systems”, *Proceedings of the Ninth International Heterogeneous Computing Workshop*, 2000.
- [100] Ranaldo, N., and Zimeo, E., “Time and Cost–Driven Scheduling of Data Parallel Tasks in Grid Workflows”, *IEEE Systems Journal*, vol. 3, no. 1, pp. 104–120, March 2009.
- [101] Ranaweera, S., Agrawal, D. P., “A Scalable Task Duplication based Scheduling Algorithm for Heterogeneous Systems”, *Proc. of the Int’l*

Conference on Parallel Processing, Toronto, Canada, pp. 383–390, 2000.

- [102] Ranaweera, S., and Agrawal, D. P., “A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems”, *Proc. of 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 445–450, Cancun, Mexico, May 2000.
- [103] Reeves, C. R., “Modern Heuristic Techniques for Combinatorial Problems”, *John Wiley & Sons*, McGraw–Hill International (UK) Limited, 1995.
- [104] Ritchie, G., and Levine, J., “A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments”, *Proc. of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*, Dec, 2004.
- [105] Sadfi, C., and Ouarda, Y., “Parallel Machine Scheduling Problem with Availability Constraints”, *Proc. of the 9th Int'l Workshop Project Management and Scheduling (PMS '04)*, 2004.
- [106] Sakellariou, R., and Zhao, H., “A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems”, *Proc. of 18th IEEE Int'l Parallel and Distributed Processing Symposium*, 2004.
- [107] Sakellariou, R., and Zhao, H., “A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems”, *Journal of Scientific Programming*, vol. 12, no. 4, pp. 253–262, 2004.
- [108] Sakellariou, R., Zhao, H., Tsiakkouri, E., and Dikaiakos, M. D., “Scheduling Workflows with Budget Constraints”, *Sergei Gorlatch and Marco Danelutto (Eds.): Integrated Research in GRID Computing*

(CoerGRID Integration Workshop 2005), Springer–Verlag, CoerGRID Series, pp. 189–202, 2007.

- [109] Sample, N., Keyani, P., and Wiederhold, G., “Scheduling Under Uncertainty: Planning for the Ubiquitous Grid”, *Proc. of the 5th Int’l Conference on Coordination Models and Languages, Lecture Notes In Computer Science*; vol. 2315, pp. 300–316, York, UK, April 2002.
- [110] Sanlaville, E., and Schidt, G., “Machine Scheduling with Availability Constraints”, *Acta Informatica*, vol. 35, no. 9, pp. 795–811, Sept. 1998.
- [111] Schopf, J., “Ten Actions When SuperScheduling”, *Document of Scheduling Working Group, Global Grid Forum*, July 2001. <http://www.ggf.org/documents/GFD.4.pdf>
- [112] SETI@Home, <http://setiathome.ssl.berkeley.edu/>
- [113] Shi, Z., and Dongarra, J. J., “Scheduling Workflow Applications on Processors with Different Capabilities”, *Future Generation Computer Systems*, vol. 22, no. 6, pp. 665–675, Elsevier, 2005.
- [114] Silva, D. P., Cirne, W., and Brasileiro, F. V., “Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids”, *Proc. of Euro-Par 2003*, pp.169–180, Klagenfurt, Austria, August 2003.
- [115] Smith, S. P., “An Efficient Method to Maintain Resource Availability Information for Scheduling Applications”, *Proc. IEEE Int’l Conf. Robotics and Automation (ICRA ’92)*, vol. 2, pp. 1214–1219, May 1992.
- [116] Song, S., Kwok, Y. K., and Hwang, K., “Security–Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling”, *Proc. of the 19th IEEE Int’l Parallel and Distributed Processing Symposium*, 2005.

- [117] Subramani, V., Kettimuthu, R., Srinivasan, S., and Sadayappan, P., "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests", *Proc. of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002)*, pp. 359–366, Edinburgh, Scotland, July 2002.
- [118] Sun, M., Zeng, G., Yuan, L., and Wang, W., "A Trust–Oriented Heuristic Scheduling Algorithm for Grid Computing", *Malyshkin, V.E. (ed.) PaCT 2007, LNCS*, vol. 4671, pp. 608–614. Springer, Heidelberg, 2007.
- [119] Sun, Xian–He, and Ming, W., "GHS: A performance prediction and task scheduling system for Grid computing", *Proc. of IEEE Int'l Parallel and Distributed Processing Symposium*, Nice, France, 2003.
- [120] Superlink@Technion, <http://cbl-link02.cs.technion.ac.il/superlinkattechnion/>
- [121] The Austrian Grid Consortium, <http://www.austriangrid.at>
- [122] The Global Grid Forum (GGF), <http://www.gridforum.org/>
- [123] The Tele Science Project, <http://www.sdsc.edu/>
- [124] Topcuoglu, H., Hariri, S., and Wu, M. Y., "Performance–Effective and Low–Complexity Task Scheduling for Heterogeneous Computing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, March 2002.
- [125] Tsiakkouri, H. Z. E., Sakellariou, R., and Dikaiakos, M. D., "Scheduling Workflows with Budget Constraints", *Proceedings of the CoreGRID Workshop on Integrated research in Grid Computing*, S. Gorlatch and M. Danelutto, Eds., pp. 347–357, 2005.
- [126] Vadhiyar, S., and Dongarra, J., "A Performance Oriented Migration Framework for the Grid", *Proc. of the 3rd Int'l Symposium on Cluster*

Computing and the Grid (CCGrid'03), pp.130–139, Tokyo, Japan, May 2003.

- [127] Viswanathan, S., Bharadwaj, V., and Robertazi, T. G., “Resource Aware Distributed Scheduling Strategies for Large–Scale Computational Cluster/Grid Systems”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, Oct 2007.
- [128] Wibisono, A., Zhao, Z., Belloum, A., and Bubak, M., “A Framework for Interactive Parameter Sweep Applications”, *Lecture Notes in Computer Science*, 2008, vol. 5103, pp. 481–490, 2008.
- [129] Wieczorek, M., Hoheisel, A., and Prodan, R., “Taxonomy of the Multi Criteria Grid Workflow Scheduling Problem”, *CoreGRID Workshop*, 2007.
- [130] Wieczorek, M., Podlipnig, S., Prodan, R., and Fahringer, T., “Bi–criteria Scheduling of Scientific Workflows for the Grid”, *8th IEEE International Symposium on Cluster Computing and the Grid* (CCGRID '08), vol., no., pp. 9–16, 19–22 May 2008.
- [131] Wolski, R., Spring, N., and Hayes, J., “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing”, *Future Generation Computing Systems*, 15(5–6):757–768, 1999.
- [132] Woo, S.–H., Yang, S.–B., Kim, S.–D., and Han, T.–D., “Task Scheduling in Distributed Computing Systems with a Genetic Algorithm”, *High Performance Computing on the Information Superhighway* (HPC Asia'97), pp. 301–305, May 1997.
- [133] Wu, M., and Sun, X., “Self–adaptive Task Allocation and Scheduling of Meta–tasks in Non–dedicated Heterogeneous Computing”, *International*

J. of High Performance Computing and Networking (IJHPCN), vol. 2, pp. 186–197, 2004.

- [134] Wu, M., Shu, W., and Zhang, H., “Segmented Min–Min: A Static Mapping Algorithm for Meta–Tasks on Heterogeneous Computing Systems”, *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, pp. 375–385, Cancun, Mexico, May 2000.
- [135] Yang, T., and Gerasoulis, A., “DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951–967, 1994.
- [136] You, S. Y., Kim, H. Y., Hwang, D. H., and Kim, S. C., “Task Scheduling Algorithm in GRID Considering Heterogeneous Environment”, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04)*, pp. 240–245, Nevada, USA, June, 2004.
- [137] Yu, J., and Buyya, R., “A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms”, *Proc. of the 15th IEEE Int'l Symposium on High Performance Distributed Computing (HPDC 2006)*, Paris, France, IEEE, IEEE CS Press, June 2006.
- [138] Yu, J., and Buyya, R., “Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms”, *Scientific Programming Journal*, vol. 14, no. 1, pp. 217–230, 2006.
- [139] Yu, J., Buyya, R., and Tham, C. K., “Cost–based Scheduling of Scientific Workflow Applications on Utility Grids”, *Proc. of the 1st IEEE Int'l Conference on e–Science and Grid Computing (e–Science 2005)*, IEEE. Melbourne, Australia: IEEE CS Press, pp. 140–147, Dec. 2005.

- [140] Yu, J., Buyya, R., and Tham, C. K., "QoS-based Scheduling of Workflow Applications on Service Grids", *Proc. of the 1st IEEE Int'l Conference on e-Science and Grid Computing (e-Science'05)*, Melbourne, Australia, December 2005.
- [141] Zhang, J., and Luo, J., "An Adaptive QoS Group Guided Grid Scheduling Algorithm with Task Replicas", *Proc. of the 11th Int'l Conference on Computer Supported Cooperative Work in Design*, 2007.
- [142] Zhang, Y., Koelbel, C., and Kennedy, K., "Relative Performance of Scheduling Algorithms in Grid Environments", *7th IEEE Int'l Symposium on Cluster Computing and the Grid*, pp. 521–528, May 2007.
- [143] Zhao, H., and Sakellariou, R., "An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm", *Lecture Notes in Computer Science (Springer)*, vol. 2790, pp. 189–194, August 2003.
- [144] Zhao, H., and Sakellariou, R., "Scheduling Multiple DAGs onto Heterogeneous Systems", *15th Heterogeneous Computing Workshop (HCW'06)*, Greece, IEEE, IEEE CS Press, April 2006.
- [145] Zhu, C., Tang, X., Li, K., Han, X., Zhu, X., and Qi, X., "Integrating Trust into Grid Economic Model Scheduling Algorithm", *R. Meersman, Z. Tari (eds.) OTM 2006, LNCS*, vol. 4276, pp. 1263–1272. Springer, Heidelberg, 2006.
- [146] Zuhily, A., and Burns, A., "Exact Scheduling Analysis of Non-Accumulatively Monotonic Multiframe Tasks", *Real Time Systems Journal*, vol. 43, pp. 119–146, 2009.

VITAE

Born on March 27, 1977 at Roorkee, he did his B.E. with Honors in Computer Science & Engineering in 1999 from G. B. Pant Engineering College, Pauri, India. Later, he did his M.Tech. in Information Technology in 2005 from Punjabi University, Patiala. He served College of Engineering Roorkee, Gurukul Kangri University, Graphic Era Institute of Technology (Presently known as Graphic Era University) and Dehradun Institute of Technology and GRD Institute of Management and Technology, Dehradun at various faculty positions. Currently, he is working with College of Engineering Roorkee, India as Professor & Dean (Computing). He pursued Doctoral Research in the Department of Electronics and Computer Engineering at Indian Institute of Technology Roorkee, India from 2007 to 2011 under MHRD scholarship of Government of India. He is a member of IEEE, member of Computer Society of India. His current areas of interest include Parallel & Distributed Computing, Grid Computing, Real Time Systems and Cloud Computing.

PUBLICATIONS

Refereed International Journals:

1. Amit Agarwal and Padam Kumar, "An Effective Compaction Strategy for Bi-Criteria DAG Scheduling in Grids", *International Journal of Communication Networks and Distributed Systems (IJCND)*, Vol. 5, No. 3, pp. 331–346, Inderscience Publications, 2010.
2. Amit Agarwal and Padam Kumar, "Economical Task Scheduling Algorithm for Grid Computing Systems", *Global Journal of Computer Science and Technology*, Vol. 10, No. 11, pp. 48–53, 2010.
3. Amit Agarwal and Padam Kumar, "Multidimensional QoS Oriented Task Scheduling in Grid Environments", *International Journal of Grid Computing and Applications (IJGCA)*, Vol. 2, No.1, pp. 28–37, 2011.
4. Amit Agarwal and Padam Kumar, "An Availability-Aware QoS Oriented Task Scheduling Strategy for Grid Computing Systems", *Int. J. of Computers and Electrical Engineering*, Elsevier. (under review)

Refereed International Conferences:

5. Amit Agarwal and Padam Kumar, "Heterogeneity-Aware Task Scheduling Using Critical Path in Grid Environments", *Proceedings of International Conference on Signal Processing Systems (ICCSA)*, Singapore, vol., no., pp. 479–483, 15–17 May 2009.
6. Amit Agarwal and Padam Kumar, "Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems", *Proceedings of IEEE International Advance Computing Conference (IACC 2009)*, Patiala, vol., no., pp. 87–93, 6–7 March, 2009.

7. Amit Agarwal and Padam Kumar, "Trust-oriented Multi-objective Workflow Scheduling in Grids", *Proceedings of International Conference on Grid and Distributed Computing (GDC 2009)*, South Korea, CCIS 63, pp. 96–107, Springer–Verlag Berlin Heidelberg, 2009.
8. Amit Agarwal and Padam Kumar, "A Two-phase Bi-criteria Workflow Scheduling Algorithm in Grid Environments", *Proceedings of 17th International Conference on Advanced Computing and Communications (ADCOM 2009)*, Bangalore, pp. 168–173, 14–17 December 2009. (Accepted)
9. Amit Agarwal and Padam Kumar, "Multicriteria Scheduling for Multiple Workflows in Grids", *Proceedings of First International Conference on Advanced Computing and Communication Technologies*, India, pp. 289–291, 2011.

This Appendix introduces different methods used in simulation to analyze the performance of proposed scheduling heuristics. The performance analysis of the proposed scheduling strategies with the related scheduling heuristics is simulated in MATLAB using TORSCHÉ scheduling toolbox [SCH]. Some new functions are implemented to design grid scheduling simulation environment. Some of the functions are described below for clarification.

A.1 Random Directed Acyclic Graph (DAG) Generation

A supplementary function 'randdfg()' allows generating random directed acyclic graph (DAG). This function has been implemented by modifying 'randdfg()' function (random data flow graph generator) available in TORSCHÉ scheduling toolbox [SCH] in order to generate DAGs. This function is as follows:

Synopsis: $g = \text{randdfg}(n, m, \text{degmax}, ne)$

The first parameter 'n' is the number of tasks in the DAG. The 'm' is the number of processing nodes (or processors) available in the grid. Parameter 'degmax' restricts the upper bound of outdegree of vertices in the DAG. Parameter ne is the number of edges in the DAG. The output parameter 'g' represents the generated DAG.

A.2 Transformations from DAG to task set

The object directed acyclic graph 'g' can be transformed to the object 'taskset' as follows:

Synopsis: $T = \text{taskset}(g)$

Each node from graph 'g' will be converted to a task. Tasks properties (e.g., Processing Time, Deadline . . .), are taken from node 'UserParam' attribute. Default order of 'UserParam' attribute is:

{'ProcTime','ReleaseTime','Deadline','DueDate','Weight','Processor','UserParam'}

All edges are automatically transformed to the task precedence constrains. Their parameters are saved to the cell array in 'T.TSUserParam.EdgesParam'. The edge list from DAG 'g' can be computed as follows:

```
% Get edge list from graph g
edge_matrix=getdata(g, 'ed');
```

A.3 Finding b-level of each task in DAG

The b-level of each task node in DAG has been computed recursively from exit task to entry task using the following MATLAB code:

```

% Calculate b-level of each node in graph g
b_level=T.ProcTime;
for i=n-1:-1:1
    for j=1:ne
        if edge_matrix(j,1)==i
            temp=b_level(edge_matrix(j, 2))+edge_matrix(j, 3)+T.ProcTime(i);
            if temp>b_level(i)
                b_level(i)=temp;
            end;
        end;
    end;
end;

```

A.4 Finding t-level of each task in DAG

The t-level of each task node in DAG has been computed using the MATLAB code as follows:

```

% Calculate t-level of each node in graph g
t_level(n)=0;
for i=2:n
    for j=1:ne
        if edge_matrix(j,2)==i
            temp=t_level(edge_matrix(j, 1))+edge_matrix(j, 3)+T.ProcTime(edge_matrix(j, 1));
            if temp>t_level(i)
                t_level(i)=temp;
            end;
        end;
    end;
end;
end;

```

A.5 Structure of scheduling algorithms in the toolbox

Scheduling algorithm in TORSCHE is a MATLAB function with at least two input parameters and at least one output parameter. The first input parameter must be taskset, with tasks to be scheduled. The second one must be an instance of problem object describing the required scheduling problem in $(\alpha | \beta | \gamma)$ notation. Taskset containing resulting schedule must be the first output parameter. Common syntax of the scheduling algorithms is as follows:

Synopsis:	<code>TS = name(T, problem[, processors[, parameters]])</code>
Name	command name of algorithm
TS	set of tasks with schedule inside
T	set of tasks to be scheduled
Problem	object problem describing the classification of scheduling problems
Processors	number of processors for which schedule is computed
Parameters	additional information for algorithms, e.g. parameters of mathematical solvers etc.

The algorithm should perform initialization of variables like 'n' (number of tasks), 'p' (vector of processing times), Then, a scheduling algorithm calculates start time of tasks (starts) and processor assignment (processor) - if required. Finally the resulting schedule is derived from the original taskset using function 'add schedule'. The structure of scheduling algorithms in the scheduling toolbox is depicted as follows [SKS06]:

```
% Structure of scheduling algorithms in the toolbox
function [TS] = schalg(T, problem)
% function description
% scheduling problem check
if ~(is(prob,'alpha','P2') && is(prob,'beta','rj,prec') && ...
is(prob,'gamma','Cmax'))
error('Can not solve this problem. ');
end
% initialization of variables
n = count(T);           %number of tasks
p = T.ProcTime         %vector of processing time
% scheduling algorithm
...
starts = ...           %assignemen of resulting start times
processor = ...       %processor assignemen
% output schedule construction
description = 'a scheduling algorithm';
TS = T;
add_schedule(TS, description, starts, p, processor);
% end of file
```

The above structure has been used to implement the proposed scheduling heuristics in the thesis. The list scheduling algorithm is implemented in the toolbox as follows:

Synopsis: `TS = listsch(T, problem, processors [,strategy])`

Strategy Strategy of the list scheduling algorithm

It is possible to define own strategy for LS algorithm according to the following model of function. Function with the same name as the optional parameter (name of strategy function) is called from List Scheduling algorithm:

Synopsis: `TS = listsch(T, problem, processors, 'OwnStrategy')`

In this case, strategy algorithm is called in each iteration of List Scheduling algorithm upon the set of unscheduled task. Strategy algorithm is a standalone function with following parameters:

Synopsis: `[TS, order] = OwnStrategy(T[, iteration, processor]);`

T set of tasks

Order index vector representing new order of tasks

Iteration actual iteration of List Scheduling algorithm

Processor selected processor

A.6 Generate Gantt Chart of the Schedule

The 'plot()' function is used to generate the Gantt chart of the output schedule TS. The syntax is as follows:

Synopsis: `plot (TS)` *% TS is a set of tasks with schedule inside*

REFERENCES:

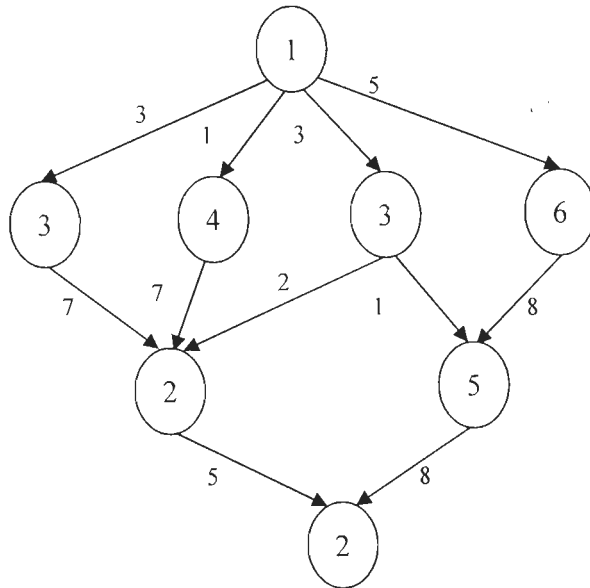
[SCH] Scheduling Toolbox, <http://rttime.felk.cvut.cz/scheduling-toolbox/>).

[SKS06] Šůcha, P., Kutil, M., Sojka, M. and Hanza'lek, Z., "TORSCHÉ Scheduling Toolbox for Matlab", *IEEE International Symposium on Computer-Aided Control Systems Design*, Germany, 2006.

In Chapter 1, Figure 1.1 depicts a directed acyclic graph of eight task nodes and eleven communication edges (grid application model). The edge weights are represented with amount (volume) of data (in Kbytes) being transferred between the concerned tasks. Figure 4.2 illustrates a GRID of four resources connected arbitrary. The solid lines show the edges among the resources whereas dotted lines represent that there is no direct communication path between the resources. Such resources may communicate using alternative path with maximum bandwidth. The communication edges of GRID are labeled with bandwidth (in Kbps). The bandwidth on indirect path can be computed by selecting a path in alternative path set which is providing the maximum bandwidth.

In Figure 4.2, resource p_1 and resource p_3 may communicate through $p_1-p_2-p_3$ yielding the maximum available bandwidth between p_1 and p_3 as 100 Kbps (minimum bandwidth of a link on the path). Similarly, the bandwidth between p_3 and p_4 may be computed by selecting a path $p_4-p_1-p_2-p_3$ which

yields a bandwidth of 100 Kbps between these resources. The mean communication rate (bandwidth) for this grid can be computed as an average bandwidth of grid. The mean communication rate is 100 Kbps for the grid depicted in Figure 4.2. Therefore, the DAG depicted in Figure 1.1 can be represented as:



The nodes are labeled with mean computation time (computed in Table 4.3) and edges are labeled as mean communication time (computed by dividing data volume by mean bandwidth). With these values, the b-level can be computed using Equation (3.7) as shown in Table 4.3. The priority task sequence generated using highest b-level is $n_1-n_5-n_4-n_3-n_2-n_7-n_6-n_8$. The tasks are scheduled over the capable resources in order of task sequence using duplication based strategy.

Figure 4.11 (a) shows the primary schedule obtained by duplication based scheduling strategy. The makespan of this schedule is 16 ms (milliseconds). The economic cost can be computed using Equation (4.1) and Equation (4.2).

For Figure 4.11(a), various processor busy times are (see Section 4.2.4):

$$PBT(p_1) = 16, PBT(p_2) = 3, PBT(p_3) = 8, PBT(p_4) = 3;$$

Economic cost can be computed for the primary schedule as:

$$EC = (16 \times 220 \times 1 + 3 \times 350 \times 2.5 + 8 \times 450 \times 3 + 3 \times 310 \times 2) / 1000 = 18.81 \text{ g\$}$$

Similarly, the economic cost after removal of useless duplications from the primary schedule in Figure 4.11(b) would be:

$$EC = (16 \times 220 \times 1 + 2 \times 350 \times 2.5 + 8 \times 450 \times 3 + 2 \times 310 \times 2) / 1000 = 17.31 \text{ g\$}$$

Further, the economic cost can be computed after removal of unproductive sub-schedules (if any) from the primary schedule in Figure 4.11(c) as:

$$EC = (16 \times 220 \times 1 + 8 \times 450 \times 3) / 1000 = 14.32 \text{ g\$}$$

In Figure 4.11 (d), the primary schedule length may be relaxed to 18 ms providing 2 ms sliding constraint (10% of makespan (16 ms)). The secondary schedule is showing a makespan of 17 ms while the tasks from resource p_3 can be migrated to p_4 . The economic cost of this schedule can be computed as:

$$PBT(p_1) = 17, PBT(p_2) = 0, PBT(p_3) = 0, PBT(p_4) = 9;$$

$$EC = (17 \times 220 \times 1 + 9 \times 310 \times 2) / 1000 = 9.32 \text{ g\$}$$

The tasks ($n_1-n_5-n_7-n_8$) show the critical path in the DAG. The NSL is computed using Equation (4.4) where denominator is the sum of minimum execution costs of tasks on critical path (i.e., 11 as per Table 4.3). therefore, NSL is as follows:

$$NSL = 17 / 11 = 1.55;$$

Now, the effective schedule cost using Equation (4.6) can be computed as:

$$ESC = 1.55 \times 9.32 = 14.4 \text{ g\$}$$

Appendix-C

MATLAB Code of AQUA Algorithm

In chapter 5, we have proposed a QoS based scheduling heuristic for scheduling independent task applications in the grid computing systems. The proposed approach (AQUA) has been simulated in MATLAB for validating with the existing QGMM approach. The code has been presented below:

Function

QGMM1(x,network_bandwidth_avail,network_bandwidth_demand,node_avail,task_avail_demand,TC,m,n)

% m is the number of task classes and n is the number of computing node, x is the %age of %dedicated nodes in the grid

```
node_avail_time=zeros(1,n);      % initialize nodes to zeros
nc=unifrnd(0.1,1,n);             % resource computing capacity
for i=1:m
    for j=1:n
        ETC(i,j)=ceil(TC(i)/nc(j));
    end;
end;

r=floor(n*100/x);                % x=25% nodes are dedicated
for i=1:r
    node_avail(i*100/x)=1;
    network_bandwidth_avail(i*100/x)=1000;
end;
```

```

task_avail_demand_temp=task_avail_demand;
task_queue=zeros(1,m);
for i=1:m
[p loc]=max(task_avail_demand_temp);
    task_queue(i)=loc;
    task_avail_demand_temp(loc)=0;
end;

queue_high=0;
queue_low=0;
high=0;low=0;
for i=1:m
    if task_avail_demand(task_queue(i))>0.5 & network_bandwidth_demand(task_queue(i))>100
        high=high+1;
        queue_high(high)=task_queue(i);
    else
        low=low+1;
        queue_low(low)=task_queue(i);
    end;
end;

queue_high_Q=0;
queue_low_Q=0;
high_Q=0;low_Q=0;
for i=1:m
    if network_bandwidth_demand(task_queue(i))>100
        high_Q=high_Q+1;
        queue_high_Q(high_Q)=task_queue(i);
    else
        low_Q=low_Q+1;
        queue_low_Q(low_Q)=task_queue(i);
    end;
end;

```

% Compute suitable node set based on availability (AQUA Approach)

```

suitable_node=zeros(m,n);
s=sum(queue_high);
if s>0
for i=1:length(queue_high)
    p=1;
    for j=1:n
        if node_avail(j)>=task_avail_demand(queue_high(i)) &
network_bandwidth_avail(j)>network_bandwidth_demand(i)
            suitable_node(queue_high(i),p)=j;
            p=p+1;
        end;
    end;
end;
end;
s=sum(queue_low);
if s>0
for i=1:length(queue_low)
    p=1;

```

```

for j=1:n
    if node_avail(j)>=task_avail_demand(queue_low(i))
        suitable_node(queue_low(i),p)=j;
        p=p+1;
    end;
end;
end;
end;

```

% Compute suitable node set based on availability (QGMM Approach)

```

suitable_node_Q=zeros(m,n);
s=sum(queue_high_Q);
ifs>0
for i=1:length(queue_high_Q)
    p=1;
    for j=1:n
        if network_bandwidth_avail(j)>network_bandwidth_demand(i)
            suitable_node_Q(queue_high_Q(i),p)=j;
            p=p+1;
        end;
    end;
end;
end;
s=sum(queue_low_Q);
ifs>0
for i=1:length(queue_low_Q)
    p=1;
    for j=1:n
        suitable_node_Q(queue_low_Q(i),p)=j;
        p=p+1;
    end;
end;
end;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Scheduling Algorithm: AQUA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

allocate=zeros(6,m);
exp_response_time=zeros(1,high);
for i=1:high % schedule high QoS tasks first
    response_time=Inf;
    class=queue_high(i);
    node_length=nnz(suitable_node(class,1:n));
    for j=1:node_length
        k=suitable_node(class,j);
        ert=ETC(class,k)+node_avail_time(k);
        exp_response_time(class)=ert;
        if exp_response_time(class)<response_time
            response_time=exp_response_time(class);
            node=k;
        end;
    end;
end;

```

```

allocate(1,queue_high(i))=node;
allocate(2,queue_high(i))=node_avail_time(node);
node_avail_time(node)=response_time;
allocate(3,queue_high(i))=response_time-allocate(2,queue_high(i));
allocate(4,queue_high(i))=response_time;
allocate(5,queue_high(i))=node_avail(node);
allocate(6,queue_high(i))=task_avail_demand(queue_high(i));
end;

exp_response_time=zeros(1,low);
for i=1:low % schedule low QoS tasks next
    response_time=Inf;
    class=queue_low(i);
    node_length=nnz(suitable_node(class,1:n));
    for j=1:node_length
        k=suitable_node(class,j);
        ert=ETC(class,k)+node_avail_time(k);
        exp_response_time(class)=ert;
        if exp_response_time(class)<response_time
            response_time=exp_response_time(class);
            node=k;
        end;
    end;
    allocate(1,queue_low(i))=node;
    allocate(2,queue_low(i))=node_avail_time(node);
    node_avail_time(node)=response_time;
    allocate(3,queue_low(i))=response_time-allocate(2,queue_low(i));
    allocate(4,queue_low(i))=response_time;
    allocate(5,queue_low(i))=node_avail(node);
    allocate(6,queue_low(i))=task_avail_demand(queue_low(i));
end;

% End of scheduling algorithm
makespan=max(node_avail_time);

temp=0;
for i=1:n
    temp=temp+makespan-node_avail_time(i);
end;
util=1-temp/(n*makespan);

grid_avail=0; %surplus availability for AQUA
for i=1:m
    grid_avail=grid_avail+allocate(5,i)-allocate(6,i);
end;
grid_avail=grid_avail/m;

success=0;
for i=1:m
    if allocate(5,i)>allocate(6,i)
        success=success+1;
    end;
end;
end;

```

```

%% Scheduling Algorithm: QGMM

```

```

node_avail_time_Q=zeros(1,n); % initialize nodes to zeros
allocate_Q=zeros(6,m);
exp_response_time=zeros(1,high_Q);
for i=1:high_Q % schedule high QoS tasks first
    response_time=Inf;
    class=queue_high_Q(i);
    node_length=nnz(suitable_node_Q(class,1:n));
    for j=1:node_length
        k=suitable_node_Q(class,j);
        ert=ETC(class,k)+node_avail_time_Q(k);
        exp_response_time(class)=ert;
        if exp_response_time(class)<response_time
            response_time=exp_response_time(class);
            node=k;
        end;
    end;
    allocate_Q(1,queue_high_Q(i))=node;
    allocate_Q(2,queue_high_Q(i))=node_avail_time_Q(node);
    node_avail_time_Q(node)=response_time;
    allocate_Q(3,queue_high_Q(i))=response_time-allocate_Q(2,queue_high_Q(i));
    allocate_Q(4,queue_high_Q(i))=response_time;
    allocate_Q(5,queue_high_Q(i))=node_avail(node);
    allocate_Q(6,queue_high_Q(i))=task_avail_demand(queue_high_Q(i));
end;

```

```

exp_response_time=zeros(1,low_Q);
for i=1:low_Q % schedule low QoS tasks next
    response_time=Inf;
    class=queue_low_Q(i);
    node_length=nnz(suitable_node_Q(class,1:n));
    for j=1:node_length
        k=suitable_node_Q(class,j);
        ert=ETC(class,k)+node_avail_time_Q(k);
        exp_response_time(class)=ert;
        if exp_response_time(class)<response_time
            response_time=exp_response_time(class);
            node=k;
        end;
    end;
    allocate_Q(1,queue_low_Q(i))=node;
    allocate_Q(2,queue_low_Q(i))=node_avail_time_Q(node);
    node_avail_time_Q(node)=response_time;
    allocate_Q(3,queue_low_Q(i))=response_time-allocate_Q(2,queue_low_Q(i));
    allocate_Q(4,queue_low_Q(i))=response_time;
    allocate_Q(5,queue_low_Q(i))=node_avail(node);
    allocate_Q(6,queue_low_Q(i))=task_avail_demand(queue_low_Q(i));
end;

```

```

% End of scheduling algorithm

```