Meta Scheduling for Market-Oriented Grid and Utility Computing

by

Saurabh Kumar Garg

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Software Engineering
The University of Melbourne, Australia

June 2010

Meta Scheduling for Market-Oriented Grid and Utility Computing

Saurabh Kumar Garg

Supervisor: Professor Rajkumar Buyya

Abstract

Grid computing enables the sharing and aggregation of autonomous IT resources to deliver them as computing utilities to end users. The management of the Grid environment is a complex task as resources are geographically distributed, heterogeneous and autonomous in nature, and their users are self-interested. In utility-oriented Grids, users define their application requirements and compete to access the most efficient and cheapest resources. Traditional resource management systems and algorithms are based on system-centric approaches which do not take into account individual requirements and interests. To this end, market-oriented scheduling is an adequate way to solve the problem. But current market-oriented systems generally, either try to maximise one user's utility or one provider's utility. Such approaches fail to solve the problem of contention for cheap and efficient resources which may lead to unnecessary delays in job execution and underutilisation of resources.

To address these problems, this thesis proposes a market-oriented meta-scheduler called "Meta-Broker", which not only coordinates the resource demand but also allocates the best resources to users in terms of monetary and performance costs. The thesis results demonstrate that considerable cost reduction and throughput can be gained by adopting our proposed approach. The meta-broker has a semi-decentralised architecture, where only scheduling decisions are made by the meta-broker while job submission, execution and monitoring are delegated to user and provider middleware.

This thesis also investigates market-oriented meta-scheduling algorithms which aim to maximise the utility of participants. The market-oriented algorithms consider Quality of Service (QoS) requirements of multiple users to map jobs against autonomous and heterogeneous resources. This thesis also presents a novel Grid Market Exchange architecture which provides the flexibility to users in choosing their own negotiation protocol for resource trading. The key research findings and contributions of this thesis are:

- The consideration of QoS requirements of all users is necessary for maximising users' utility and utilisation of resources. The uncoordinated scheduling of applications by personalised user-brokers leads to overloading of cheap and efficient resources.
- It is important to exploit the heterogeneity between different resource sites/data centers while scheduling jobs to maximise the provider's utility. This consideration not only reduce energy cost of computing infrastructure by 33% on average, but also enhance the efficiency of resources in terms of carbon emissions.
- By considering both system metrics and market parameters, we can enable more effective scheduling which maximises the utility of both users and resource providers.

This is to certify that

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

Signature_____

Date_____

ACKNOWLEDGMENTS

This thesis is the story of a journey throughout which I received help from many people, some of them helped me from outside and others gave me unforgettable internal motivation.

First, I would like to thank my supervisor, Professor Rajkumar Buyya for his intellectual guidance and continuous encouragement which ensure the successful completion of this thesis. He always gave me freedom to think broadly and deeply into my research. The regular meetings conducted by him made me work more regularly and systematically. His endless amount of energy is really an inspiration for me to do better research.

I am thankful to CLOUDS Lab group members including Srikumar Venugopal, Chee Shin Yeo, Marcos Assuncao, Mukaddim Pathan, Marco Netto, Mustafizur Rahman, James Broberg, Suraj Pandey, Adam Barker, Christian Vecchiola, William Voorsluys, Mohsen Amini, and Rajiv Ranjan for their patience and tolerance in helping me. I am extremely grateful to Chee Shin, Marco, Srikumar and Marcos for proof-reading, discussion and their comments on my work. Their critical comments helped me to improve my writing skills.

I would also like to thank my collaborators: Pramod Kumar Konugurthi (Indian Space Research Organisation, Hyderabad, India), Arun Anandasivam (Universitt Karlsruhe, Germany), and Professor Howard Jay Siegel (Colorado State University, USA). I would like to thank Professor Rao Kotagiri for being my PhD committee member. I am grateful to Dr. Jemal Abawajy for going through my thesis and giving many serious comments. I would like to thank administrative staff members in the Computer Science and Software Engineering (CSSE) Department especially Pinoo Bharucha for her support and help.

I thank sponsors of the CLOUDS Laboratory, in particular the Australian Research Council (ARC), and Australian Department of Innovation, Industry, Science and Research (DIISR) for funding the research in this thesis. Along with CLOUDS Lab, the CSSE department and Melbourne University provide the infrastructure for my research and travel support to international conferences.

I would like to thank my friends particularly Chee Shin and XiaoFeng for tolerating my long discussions on all type of topics. I am immensely grateful to my teachers for their exemplary lives which always inspired me to become a responsible genuine intellectual. Last but never the least, I would like to thank my family for their love, patience and support at all times. Every week, my mother used to ask me when I will complete the thesis and visit home.

Saurabh Kumar Garg
Melbourne, Australia
October 2010.

CONTENTS

1 Introduction **1**
- 1.1 Grid and Utility Computing . . . 2
- 1.2 Limitations of Existing Scheduling Mechanisms . . . 3
- 1.3 Problem Statement and Objectives . . . 5
- 1.4 Proposed Solution . . . 6
- 1.5 Thesis Contributions . . . 8
- 1.6 Thesis Organisation . . . 9

2 Taxonomy of Market-Oriented Scheduling Mechanisms **13**
- 2.1 Overview of Utility Grids and Preliminaries . . . 13
- 2.2 Requirements . . . 14
  - 2.2.1 Consumer Requirements . . . 15
  - 2.2.2 Resource Provider Requirements . . . 16
  - 2.2.3 Market Exchange (ME) Requirements . . . 17
- 2.3 Utility Grid Infrastructural Components . . . 19
- 2.4 Taxonomy of Market-Oriented Scheduling . . . 21
  - 2.4.1 Market Model . . . 21
  - 2.4.2 Allocation Decision . . . 26
  - 2.4.3 Participant Focus . . . 26
  - 2.4.4 Application Type . . . 27
  - 2.4.5 Allocation Objective . . . 27
- 2.5 Survey of Grid Resource Management Systems . . . 28
  - 2.5.1 Survey of Market-Oriented Systems . . . 28
  - 2.5.2 System-Oriented Schedulers . . . 36
- 2.6 Discussion and Gap analysis . . . 41
  - 2.6.1 Scheduling Mechanisms . . . 42
  - 2.6.2 Market-Oriented Systems . . . 43
- 2.7 Summary . . . 44

3 Market-Oriented Meta-Scheduler Architecture **45**
- 3.1 Motivation . . . 45
- 3.2 Meta-Broker Architecture . . . 45
  - 3.2.1 Architectural Components . . . 46
- 3.3 Resource Allocation by the Meta-Broker . . . 50
- 3.4 Meta-Broker's Internal Control Flow . . . 50
- 3.5 Comparison between Personalised and Meta-Broker . . . 51
- 3.6 Performance Results . . . 53
- 3.7 Summary . . . 54

4 Meta-Scheduling to Minimise User Spending **55**
4.1 Problem Definition . . . 55
4.1.1 Problem Formulation . . . 56
4.2 Proposed Algorithms . . . 57
4.2.1 Linear Programming-based Algorithm for Scheduling MGN Jobs 57
4.2.2 Linear Programming-based Algorithm for Scheduling SGN Jobs . 58
4.3 Performance Evaluation . . . 62
4.3.1 Simulation Methodology . . . 62
4.3.2 Performance Results . . . 65
4.4 Related Work . . . 68
4.5 Summary . . . 69

5 Meta-Scheduling to Minimise Time and Cost for Users **71**
5.1 Motivation . . . 71
5.2 Meta-Broker System . . . 71
5.3 Meta-Scheduling Algorithms . . . 72
5.3.1 Problem Statement . . . 72
5.3.2 Min-Min Cost Time Trade-off (MinCTT) Heuristics . . . 74
5.3.3 Max-Min Cost Time Trade-off (MaxCTT Heuristics) . . . 74
5.3.4 Sufferage Cost Time Tradeoff (SuffCTT Heuristics) . . . 75
5.3.5 Time Complexity . . . 75
5.4 Simulation Setup . . . 76
5.5 Analysis of Results . . . 79
5.5.1 CASE 1: Trade-off Factor Set by Meta-broker . . . 79
5.5.2 CASE 2: Trade-off Factor Set by User . . . 81
5.6 Related Work . . . 86
5.7 Summary . . . 88

6 Meta-Scheduling to Maximise Provider's Utility **91**
6.1 Motivation . . . 91
6.2 Meta-scheduling Model . . . 93
6.2.1 Data Center Energy Model . . . 94
6.2.2 Relation between Execution Time and CPU Frequency . . . 96
6.2.3 Problem Description . . . 96
6.3 Meta-Scheduling Policies . . . 99
6.3.1 Mapping Phase (Across Many Data Centers) . . . 99
6.3.2 Scheduling Phase (Within a Data Center) . . . 101
6.3.3 Lower Bound and Upper Bound . . . 102
6.4 Performance Evaluation . . . 104
6.5 Analysis of Results . . . 107
6.5.1 Evaluation without Data Transfer Cost . . . 107
6.5.2 Evaluation with Data Transfer Cost . . . 116
6.6 Summary . . . 118

7 Meta-Scheduling to Enhance All Grid Players' Utility **121**
7.1 Motivation . . . 121
7.2 System Model . . . 122
7.3 Double Auction-Inspired Meta-scheduling (DAM) . . . 122
7.3.1 Valuation Mechanism . . . 124
7.3.2 The Meta-Scheduling Algorithm . . . 126
7.3.3 Queueing Theory Based Model for Meta-scheduling . . . 128
7.4 Performance Evaluation . . . 134
7.4.1 Experimental Configuration . . . 134
7.4.2 Analysis of Results . . . 137
7.5 Summary . . . 142

8 Market Exchange and Meta-Broker Implementation **143**
8.1 Motivation . . . 143
8.2 Market Exchange Requirements . . . 144
8.2.1 Infrastructure Requirements . . . 144
8.2.2 Market Requirements . . . 145
8.3 Mandi Architecture and Design . . . 146
8.3.1 Design Considerations and Solutions . . . 146
8.3.2 Architectural Components . . . 147
8.3.3 High Level Description . . . 149
8.3.4 User Interaction Phases . . . 150
8.3.5 Implementation Details . . . 151
8.4 Prototype and Performance Evaluation . . . 155
8.4.1 System Details . . . 156
8.4.2 Performance Evaluation . . . 156
8.4.3 Discussion . . . 159
8.5 Summary . . . 160

9 Conclusions and Future Directions **161**
9.1 Summary . . . 161
9.2 Lessons Learned and Significance . . . 162
9.3 Future Directions . . . 165
9.3.1 Resources with Different Pricing Models . . . 165
9.3.2 Scheduling with Pre-emption . . . 165
9.3.3 Network and Data-Aware Application Meta-scheduling . . . 166
9.3.4 SLA based Meta-Scheduling in Cloud Computing Environments . 166
9.3.5 Energy-efficient Meta-Scheduling . . . 166
9.3.6 Scalable Meta-Broker Architecture . . . 167
9.3.7 Mixed Application Model with QoS Requirements . . . 167

References **168**

LIST OF FIGURES

1.1 A view of market-oriented Grid pushing Grid into mainstream computing 3
1.2 Market-Oriented Meta-Scheduling Scenario 7
1.3 Thesis Organisation 10

2.1 A Grid Market Exchange Managing Self-Interested Entities (Providers and Consumers) 14
2.2 Utility Grid Component 20
2.3 Taxonomy of Market-based Mechanisms 22

3.1 Meta-Broker Architectural Components 46
3.2 Bag-of-Task Application Scheduling 48
3.3 Parallel Application Scheduling 48
3.4 Meta-Broker Protocol 51
3.5 Effect of Tight Deadline on Users 52
3.6 Effect of Medium Deadline on Users 53
3.7 Effect of Relax Deadline on Users 53

4.1 Effect on User Applications with Tight Deadline 66
4.2 Effect on User Applications with Medium Deadline 66
4.3 Effect on User applications with Relaxed Deadline 67
4.4 Comparison of Number of Iterations in HGA and LPGA 67

5.1 Overall Average Cost of Execution 79
5.2 Overall Makespan 80
5.3 Different ETC and Resource Pricing Configurations 82
5.4 User Application Distribution on Resources in Different Configurations . 84
5.5 Effect of Scheduling Interval in HICC Configuration 85
5.6 Effect of DTC on Cost and Time 85
5.7 Effect of Change in Application Submitted on Cost 86
5.8 Effect of Change in Application Submitted on Overall Makespan 87

6.1 Computer Power Consumption Index (Source: [73]) 92
6.2 Effect of Mapping Policy and DVS 108
6.3 Exploiting Local Minima in DVS 110
6.4 Comparison of Lower Bound and Upper Bound 111
6.5 Impact of Urgency and Arrival Rate of Applications 112
6.6 Impact of Carbon Emission Rate 113
6.7 Impact of Electricity Price 115
6.8 Impact of Data Center Efficiency 116
6.9 Impact of Data Transfer Cost 117

7.1 Double Auction based Meta-Scheduling Protocol 123

7.2 Comparison of Multiplicative and Additive forms of Valuation Metrics . . 125
7.3 Available Queue Slots 126
7.4 Queuing Theory View of Meta-scheduling 129
7.5 Comparison of DAM with analytical results 137
7.6 Benefit for Users 139
7.7 Variation in Load across Resources 141

8.1 Mandi Architecture 148
8.2 Mandi Class Design Diagram 152
8.3 Registration Process 154
8.4 Scheduling Sequence 154
8.5 Reservation Process 155
8.6 The Topology of Testbed 157
8.7 Performance of Mandi for 50,000 clearance requests 158

LIST OF TABLES

2.1 Market-Oriented Scheduling Systems 32
2.2 System-Oriented Schedulers 39

4.1 Total Cost Spent by Users for MGN Jobs 65

5.1 Lublin Workload Model Parameter Values 77
5.2 Simulated EDG Testbed Resources. 77

6.1 Comparison of Related Work 93
6.2 Parameters of a Data Center i 96
6.3 Characteristics of Data Centers 105
6.4 Summary of Heuristics with Comparison Results 118

7.1 Workload Characteristics 135
7.2 Simulated EDG Testbed Resources 136

8.1 Overhead due to Interactions of Mandi 159

Chapter 1

Introduction

"As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of ***computer utilities****, which, like present electric and telephone utilities, will service individual homes and offices across the country".*(Kleinrock 1969)

In 1969, Leonard Kleinrock, the chief scientist of the original Advanced Research Project Agency (ARPA) project, predicted that the use of computer networks would be so wide spread that computing would be used as a "utility" [99]. From 1969 till today, Information and Communication Technology (ICT) has made key advances in various areas to make this vision a reality [27]. The advances in high performance processors (such as multi-core technology) and networked computing environments have transformed computing to a model consisting of services that can be commoditised and delivered in a manner similar to utilities such as water, electricity, gas, and telephony [26]. In the utility model, users can access services on-demand based on their requirements, without regard to where the services are hosted.

The utility computing model can offer great opportunities and benefits to Information Technology (IT) users [65]. The most prominent advantage of the utility computing is the reduction of IT-related operational costs and complexities. Users from different domains, such as health care, life sciences, software development, digital media, manufacturing and petroleum, no longer need to invest heavily or encounter difficulties in building and maintaining IT infrastructure. Furthermore, the utility computing can benefit many small businesses, which lack the working capital to purchase IT infrastructure required to meet their business objectives. Hence, instead of maintaining expensive infrastructure themselves, companies will be able to submit their tasks to utility computing providers. In short, utility computing promises to provide businesses with greater flexibility and resilience, and at the same time, more efficient utilisation of resources at lower operating and maintenance costs.

Many computing paradigms [27] such as the Web, Data Centers, Service Computing/Web Services, Grid Computing, P2P Computing and Cloud Computing have emerged to support such a utility model for IT services. Among these, Grid computing is one of the most promising paradigms, which can supply computing as a utility, and thus provide an on-demand service. Grids enable the sharing, selection, and aggregation of a wide variety of autonomous and geographically distributed resources including supercomputers, storage systems, data sources, and specialised devices. However, the management of resources and applications; scheduling in such a large-scale distributed environment is a complex undertaking. Grid users and resource providers have different goals, objectives, strategies, and requirements, which need to be matched. Therefore, this thesis addresses these resource management challenges in Grids using market-oriented approach.

In the following section, a brief introduction to Grid computing is presented, describing the motivation and challenges, in the context of resource allocation, that lead to the evolution of utility-oriented Grids. The resource allocation in utility Grids will be discussed by identifying gaps in the existing research. In the end of this chapter, the core contributions of this thesis are summarised and an outline of its organisation is presented.

1.1 Grid and Utility Computing

Grid computing infrastructure promises to provide us with the ability to dynamically link together resources, as an ensemble to support the execution of large-scale, resource-intensive, and distributed applications in science, engineering, and commerce [65]. Thus, it offers a common, shared platform on which applications can be executed in a flexible way, facilitating more effective utilisation of computing resources across conventional organisational and data center boundaries. Furthermore, this platform can be easily managed and dynamically scaled using new on-demand utility services provided by a range of industry suppliers [22]. The distributed infrastructure, such as TeraGrid [29] and LHC-Grid [82], are the examples of Grids that are deployed around the world in both academic and commercial settings.

The widespread interest in Grid computing from commercial organisations in recent times is pushing it towards the mainstream, thus enhancing Grid services and enabling them to become valuable economic commodities. Most of the leading IT service companies have announced initiatives in the area of utility computing services under different business names. For example, Sun Microsoft has proposed the “Sun Grid Engine” [71], while IBM is offering “On-demand business” [81]. Many Resource Management Systems (RMSs) and meta-schedulers, such as GridWay [85] and Gridbus broker [163], have been extended to include market-oriented scheduling and resource allocation, and thus contributing to the advancement of utility Grids.

The utility Grid environment can be considered as a market where competition occurs between both consumers and providers. The main participants of this market are:

- Resource consumers who have varying resource requirements to run applications.
- Resource providers who lease resources such as CPUs and storage on-demand, in exchange for some reward.

Each of these participants is interested in maximising his/her own utility. The creation of utility Grids requires the integration of scalable system architecture, resource management and scheduling, and market models, as shown in Figure 1.1. In order to enable the consumers to participate in the utility Grid, mechanisms for bidding and cost-minimisation are required [39][122]. Similarly, many admission control and pricing polices are required to enable resource providers to participate in the market [139][3].

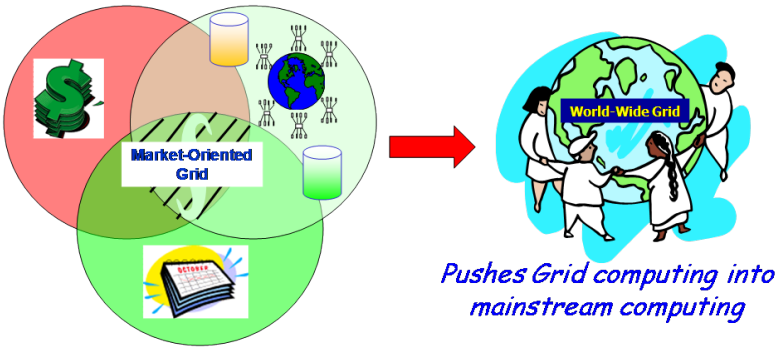


Figure 1.1: A view of market-oriented Grid pushing Grid into mainstream computing

Several researchers have satisfied these requirements by providing infrastructure such as Grid Information Service (GIS) [41] for the development of pricing [148] and market mechanisms [127][121]. There are challenges at infrastructure level, but the key issue is how to match buyers' application requirements with sellers' resources. More details on the resource management and scheduling are discussed along with their limitations in the following section.

1.2 Limitations of Existing Scheduling Mechanisms

Resource management is a central and most challenging task in Grids. The basic responsibility of a RMS is to accept requests from users, match these requests to available resources for which the user has access and then schedule the execution using the matched resources. Grid resources include compute cycles, network bandwidth and storage systems. In Grids, users generally rely on meta-schedulers [85] [11] or personal brokers [163], to ensure the satisfaction of their requirements by the discovery and reservation of suitable resources across multiple autonomous sites.

The efficient resource management and scheduling in Grids continue to be a complex and difficult undertaking [22]. One of the problems is dealing with geographically distributed autonomous resources with different usage policies, cost models, varying loads, and availability patterns. The Grid service providers (resource owners) and consumers (resource users) have different goals, objectives, strategies, and requirements. Resource sharing becomes further complicated in utility Grids due to the self-interested nature of users. The traditional resource management techniques for Grids focus on system-centric metrics such as maximising throughput, minimising mean waiting time and slowdown. In contrast, most of these resource management techniques need to be extended to include the competitive nature of participants with conflicting Quality of Service (QoS) requirements in utility Grids.

The participants in a utility Grid are more interested in maximising their profit rather than optimising performance metrics such as global resource utilisation, without receiving any direct reward [16]. Thus, in a shared infrastructure such as Grids, the self-interested nature of users can lead to problems, such as "Tragedy of Commons" [77], where every user acquires as many resources as possible because there is no incentive for users to back off during times of high demand. The self-interested users, in turn, over exploit the service by degrading the system's ability to deliver the required service to all users. Therefore, in Grids, resource management and scheduling need to be market-oriented, which can regulate the supply and demand of resources at peak usage time. The resource managers and schedulers should also provide feedback in terms of economic incentives for both Grid consumers and providers, and promote QoS-based resource allocation mechanisms that differentiate resource requests based on their utilities.

In order to meet the above requirements, most researchers have proposed either mechanisms considering a one-sided market (involving one provider or consumer) such as Vickery, First Price and Second Price auction [16]; or a two-sided market (involving multiple providers and consumers) such as Double auction. Most of these mechanisms are inspired by the trading methodologies used in the real world markets [21], for instance, auction mechanisms. The one-sided markets mostly aim to maximise the utility of one participant, while the two-sided mechanisms aim to maximise the utility of both participants (i.e. consumers and providers).

The one-sided mechanisms overcome many limitations of traditional scheduling approaches by making users pay for their usage, but there are many unresolved issues in this approach. In general, the one-sided markets favour monopolistic sellers or monopolistic buyers. They assume a completely decentralised system where each user can negotiate with any provider on a one-to-one basis. With respect to the applicability of markets in Grids, Nakai et a. [117] have pointed out that a completely decentralised system is infeasible, and can lead to inefficiency and even underutilisation of resources. The competition

among concurrent users with different requirements can exacerbate the contention for efficient and cheap resources. This contention can cause long delays in the scheduling of user applications.

To overcome these shortcomings, market-based mechanisms such as proportional share [104] have been proposed to allocate resources to bidders (Grid users) on the basis of relative share. While these approaches distribute resources fairly and reduce the response time, they limit the ability of customers to express fine-grained preferences. Moreover, these mechanisms can only be applied within a single resource site. In addition, the self-interested and strategic nature of users makes the valuation of their requirements and resources a challenging task.

Market mechanisms, such as combinatorial [179] and Double auction [92], have been proposed to match multiple users and providers in order to maximise utility of all participants. However, the practical usage of these mechanisms has remained limited since they select the auction winner based on user valuations (bids). This can lead to the starvation of low budget user applications, and thus monopolisation of the market by the rich users. The problem is critical since one user can buy all of the efficient resources, therefore resulting in a scarcity of resources for other users. Furthermore, since the major commercial resource providers, such as Amazon [6] and Sun [71], have adopted the commodity market model, these auction mechanisms cannot be applied practically in the current scenario.

Existing mechanisms [25][174][88] in the context of commodity markets also allocate resources without taking into account the effect of one user's requirements on other users. Therefore, these mechanisms also end up in similar problems, where one accepted transaction can affect more than just the immediate resource consumer or provider.

Thus, in order to schedule applications from competitive users while maximising their utility, this thesis proposes a market-oriented meta-scheduler and mechanisms for efficient, coordinated and cost-effective allocation of distributed computational Grid resources, especially in the commodity market.

1.3 Problem Statement and Objectives

This thesis focuses on the following problem:

Designing algorithms for efficient and cost-effective scheduling of multiple applications from competing users in utility Grids to distributed and heterogeneous resources under different autonomous administrative domains.

The previous section discussed many challenges in designing mechanisms which ensure efficient usage of available resources, and satisfy the requirements of multiple users. First, users, through their personalised brokers [163], compete for cheap and efficient resources, which can lead to disproportionate distribution of load. Moreover, Grid resources

are typically controlled within self-interested autonomous administrative domains. Thus, the conflict of interests between users and providers is inevitable. This conflict between competing parties needs to be reconciled in order to maximise the utility of all the participants. Other than heterogeneity at the resource level, the challenge from the users is to meet their complex requirements, which include both monetary and performance costs. Finally, this is a combinatorial resource allocation problem which is well-known to be NP-hard [118]. Therefore, in order to handle the resource demands from competing users, the requirements of both users and resource providers need to be considered, when designing efficient and cost-effective scheduling mechanisms.

Based on the issues described above, we identified the following objectives:

- To investigate the architectural model for market-oriented meta-scheduler to coordinate resource demands.
- To design meta-scheduling algorithms and mechanisms that can reconcile resource demands from users with conflicting requirements.
- To investigate how to reduce the impact on the utiity of all participants, while considering other system-centric metrics such as response time.

1.4 Proposed Solution

To solve the problem of allocating resources to competing users, this thesis proposes a market-oriented meta-scheduler called "Meta-Broker", which coordinates the resource demand from users. The meta-broker facilitates the scheduling of multiple applications on distributed autonomous resources using market-oriented techniques. To enable this, the meta-broker (Figure 1.2) periodically interacts with user brokers and local schedulers of resource sites. The meta-broker acts as a matchmaker, an information provider, a coordinator, negotiator and aggregator. Its goal is to maximise the utilities of the participants i.e. users and resource providers. The key difference between the architecture of meta-broker and meta-schedulers such as GridWay [85] is that the meta-broker only deals with matchmaking of jobs and resources, while the actual job submission is done by user brokers. In terms of scheduling mechanisms, the key difference is that the meta-broker considers *both monetary and performance QoS* requirements from *all the applications* in order to decide resource allocations.

In utility Grids, there are three possible scenarios that can occur with regard to the mapping of multiple user applications to multiple resource sites:

1. A community of users access multiple resources in utility Grids. In this scenario, users are cooperative and providers are competitive. Thus, the meta-broker tries to maximise users' utility while satisfying their QoS requirements.

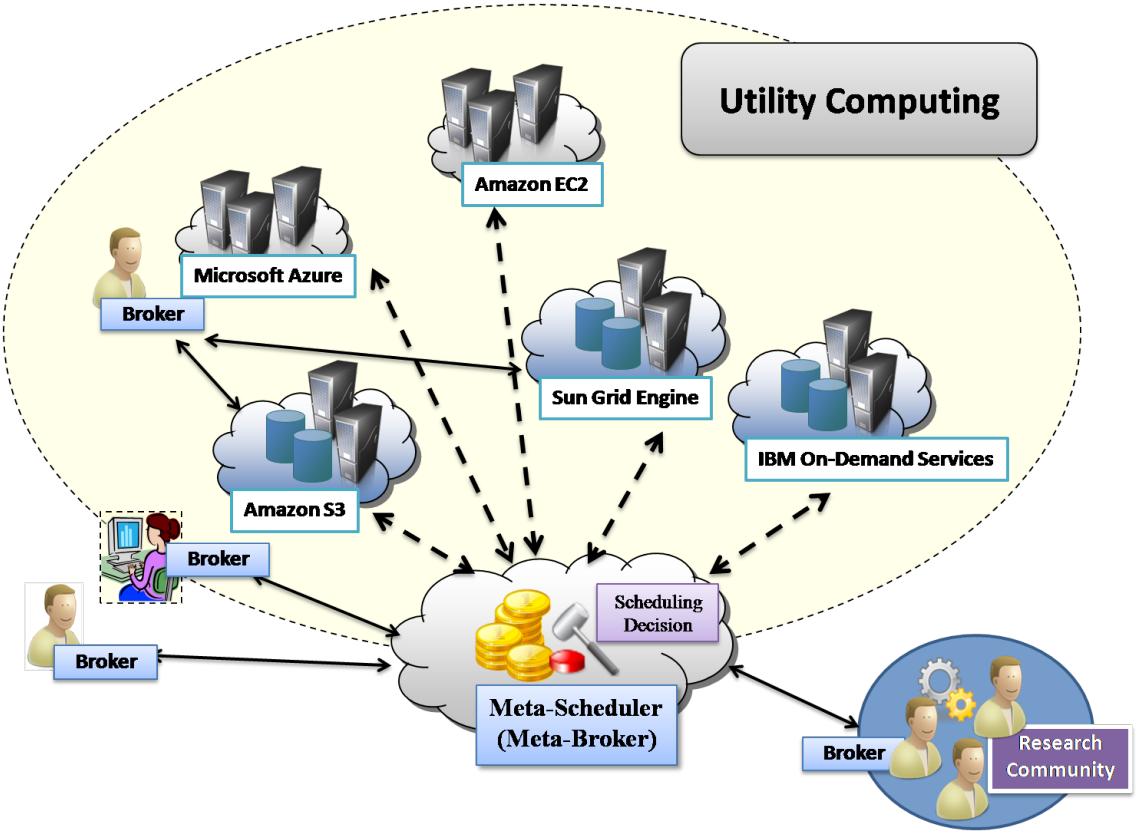


Figure 1.2: Market-Oriented Meta-Scheduling Scenario

2. Multiple users try to access resources from the same provider. In this case, users compete with each other to access the resources. The provider owns multiple resource centers, which are geographically distributed. The objective of the meta-broker is to schedule the applications on multiple resource sites so that the provider's utility can be maximised.

3. Both users and providers are competitive and want to maximise their utilities. Thus, the meta-broker aims to maximise both users' and providers' utility when mapping user applications to resources.

In this thesis, we have designed market-oriented meta-scheduling allocation mechanisms that resolves the conflicting requirements of users for each of these scenarios.

Comparing various algorithms in a utility Grid environment with different resource configurations and user requirements is difficult or almost impossible since the effect of other users in different administrative domains cannot be controlled. Therefore, to evaluate the proposed market-oriented meta-scheduling algorithms in a controlled and repeatable environment, experiments are performed on a simulated utility Grid environment using a discrete-event simulator called GridSim [24].

1.5 Thesis Contributions

This thesis contributes towards the advancement of market-based meta-scheduling of multiple user applications on resources in a utility Grid environment. The contributions are as follows:

1. This thesis provides a comprehensive taxonomy of market-based scheduling mechanisms that cover various aspects, such as market model, application model, participant focus, allocation decision and objectives. The taxonomy is intended to help researchers to make cooperative efforts towards the goal of utility-oriented Grids, by providing insights to key issues that are still outstanding. Based on the taxonomy, a survey of the most relevant market-oriented systems and traditional resource management systems is presented with a comprehensive comparison. The survey gives the insights which are helpful in extending and reusing components of existing Grid middleware. Therefore, the taxonomy and survey also highlight various research gaps to enhance the state-of-the-art of market-oriented systems in utility Grids.

2. This thesis presents the design and development of a market-oriented meta-scheduler i.e. "Meta-Broker", which coordinates concurrent users and performs scheduling on multiple resources. It investigates the benefit of central coordination over completely decentralised scheduling by personalised user brokers.

3. This thesis models the meta-scheduling problem to maximise users' utility using a Linear Programming/Integer Programming (LP/IP) model. It investigates the problem for two different types of users' utility functions: a) minimisation of monetary cost, and b) simultaneously minimisation of the monetary cost and response time. In the first case, this thesis designed a hybrid meta-scheduling algorithm, which combines the advantages of LP and genetic algorithms, for searching the cheapest resource allocation. In the second case, users need to manage the trade-off between the monetary and performance costs. Thus, this thesis analyses the problem using a trade-off metric, and presented the heuristics to minimise both monetary cost and response time.

4. This thesis investigates how to maximise provider's profit (utility) while reducing the carbon emissions by energy-aware meta-scheduling of applications on globally distributed data centers. To achieve this objective, this thesis identifies various essential factors such as energy cost, CO_2 emission rate, compute-intensive workload, and CPU power efficiency. A novel analytical model, with dual objectives of profit and carbon emission, is presented based on these factors. The near-optimal energy-efficient scheduling policies not only minimise the CO_2 emissions and maximise

the profit of the provider, but also can be readily implemented without much infrastructure changes, such as the relocation of existing data centers.

5. This thesis investigates how market-based meta-scheduling can be used to maximise the utility of both the participants of Grid. It analyses the problem in terms of system metrics, such as slowdown and waiting time, by using a queuing theory based analytical model. It then presents a meta-scheduling mechanism which takes advantages of both market based and system based approaches, in order to maximise both users' and resource providers' utility. This is demonstrated via valuation metrics that commodify the resource share available and users' application requirements so that they can be compared and matched.

6. A single Grid market protocol for physical resources such as CPU and memory is insufficient to ensure the successful adoption of Grid computing across organisational boundaries. Instead a set or catalogue of different market places is needed to satisfy the diverse needs of different market segments. Thus, this thesis presents the design and architecture of a market exchange, and its requirements to ensure the simultaneous existence of various negotiation protocols similar to a real market.

1.6 Thesis Organisation

The rest of the thesis is organised (Figure 1.3) as follows: Chapter 2 presents the survey and taxonomy of market-oriented scheduling systems and mechanisms. This chapter offers the literature background for the remaining part of this thesis by highlighting research gaps in the utility Grid area. Chapter 3 describes the architecture of our market-oriented meta-scheduler (*aka* Meta-Broker) with the details of our system model which is used in this thesis. In this chapter, we evaluate our proposed meta-broker architecture with personalised user brokers to show its benefits. Chapter 4 and 5 propose the meta-scheduling algorithms in the context of the first scenario, i.e., to minimise users' expenses. Chapter 4 presents the meta-scheduling algorithm to minimise the user spending in execution of jobs with QoS constraints such as budget and deadline. Chapter 5 analyses the trade-off between performance and monetary costs while scheduling the user applications in a utility Grid environment. Chapter 6 describes how to model the meta-scheduling problem in the second scenario. It presents meta-scheduling algorithms that maximise the provider's utility by reducing the energy cost of the infrastructure. This chapter also analyses the performance of our algorithms in terms of its impact on global carbon emissions and provider's profit. Then, Chapter 7 discusses the meta-scheduling problem in the context of the third scenario. In this chapter, we present an auction based meta-scheduling algorithm which benefits both users and providers by maximising the number of scheduled

applications and balancing resource load respectively. Chapter 8 presents the implementation of our meta-broker, which is realised within a market exchange called "Mandi". Chapter 9 concludes and provides directions for future work. The publications resulted from this thesis are:

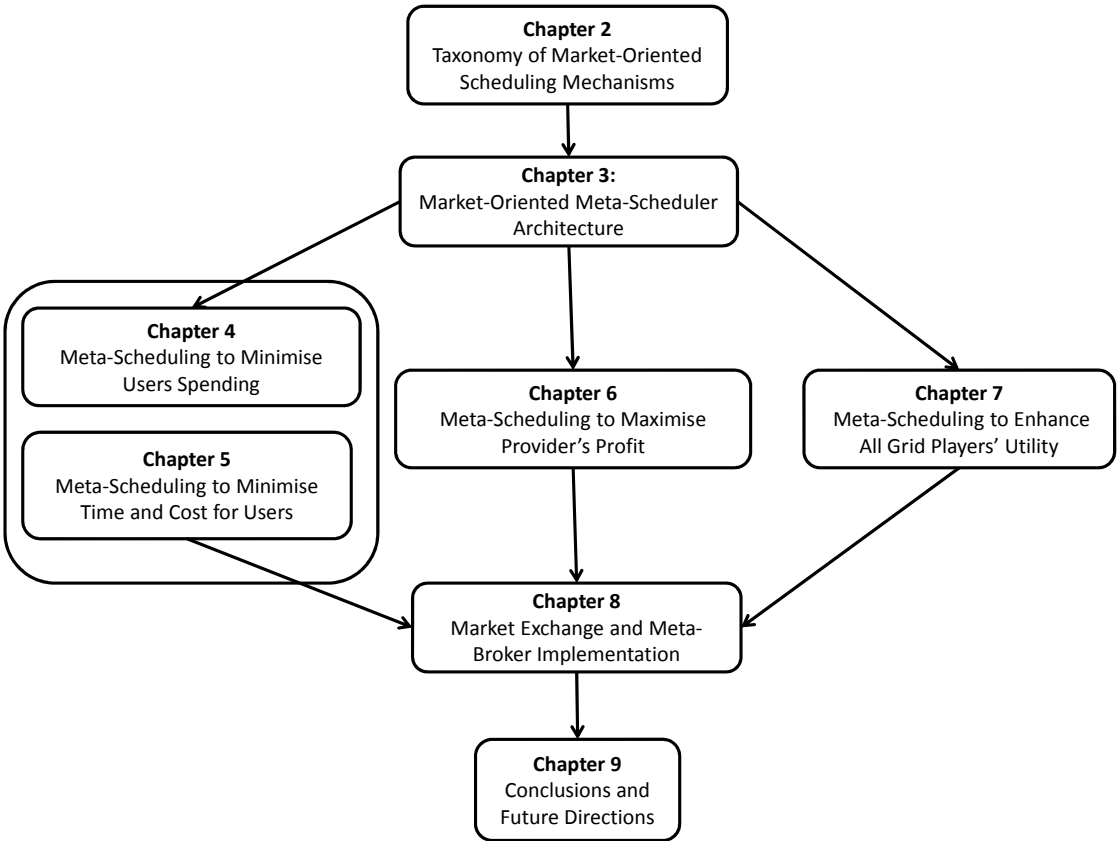


Figure 1.3: Thesis Organisation

- Chapter 2 is derived from the following publication:
  - **Saurabh Kumar Garg** and Rajkumar Buyya, Market-Oriented Resource Management and Scheduling: A Taxonomy and Survey, Cooperative Networking, M. S. Obaidat and S. Misra (eds), Wiley Press, New York, USA, 2011.
- Chapter 3 and 4 is derived from the following publications:
  - **Saurabh Kumar Garg**, Pramod Konugurthi, and Rajkumar Buyya, A Linear Programming Driven Genetic Algorithm for Meta-Scheduling on Utility Grids, Proceedings of the 16th International Conference on Advanced Computing and Communication (ADCOM 2008), Chennai, India, 2008. ***(Received "Best Paper" Award)***

  - **Saurabh Kumar Garg**, Pramod Konugurthi, and Rajkumar Buyya, Genetic Algorithms-based Heuristics for Meta-Scheduling on Utility Grids, International Journal of Parallel, Emergent and Distributed Systems, ISSN: 1744-5760, Taylor & Francis Publication, UK, 2011.

- Chapter 5 is derived from the following publications:

  - **Saurabh Kumar Garg**, Rajkumar Buyya, Howard Jay Siegel, Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management, Proceedings of the 32th Australasian Computer Science Conference (ACSC 2009), Wellington, New Zealand, 2009. ***(Received "Best Paper" and "Best Student Paper" Award)***
  - **Saurabh Kumar Garg**, Rajkumar Buyya, and Howard Jay Siegel, Time and Cost Trade-off Management for Scheduling Parallel Applications on Utility Grids, Future Generation Computer Systems, vol-26, no. 8, ISSN: 0167-739X, doi:10.1016/j.future.2009.07.003, Elsevier Science, Amsterdam, The Netherlands, 2010.

- Chapter 6 is derived from the following publications:

  - **Saurabh Kumar Garg** and Rajkumar Buyya, Exploiting Heterogeneity in Grid Computing for Energy-Efficient Resource Allocation, Proceedings of the 17th International Conference on Advanced Computing and Communications (ADCOM 2009), Bengaluru, India, 2009.
  - **Saurabh Kumar Garg**, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya, Environment-Conscious Scheduling of HPC Applications on Distributed Cloud-oriented Data Centers, Journal of Parallel and Distributed Computing (JPDC), ISSN: 0743-7315, doi:10.1016/j.jpdc.2010.04.004, Elsevier Press, Amsterdam, The Netherlands, 2010.

- Chapter 7 is derived from the following publications:

  - **Saurabh Kumar Garg**, Srikumar Venugopal, James Broberg and Rajkumar Buyya, Double Auction based Meta-Scheduling of Parallel Applications on Global GridsTechnical Report, CLOUDS-TR-2009-9, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Sept. 3, 2009.

- Chapter 8 is derived from the following publication:

– **Saurabh Kumar Garg**, Christian Vecchiola, and Rajkumar Buyya, Mandi: A Market Exchange for Trading Utility Computing Services, Technical Report, CLOUDS-TR-2009-13, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Nov. 17, 2009.

Chapter 2

Taxonomy of Market-Oriented Scheduling Mechanisms

The shift of Grids from providing compute power on a sharing basis to commercial purposes, even though it has not fully unfolded and it is still mostly limited to research, has led to various technical advancements which have paved a way to make utility Grids a reality. These advancements favour the application of market-oriented mechanisms for Grid systems by providing various pre-requisites, such as hardware virtualisation, on the technical and economic sides. This chapter summarises the recent advances toward the vision of utility Grids. First, it specifies all the requirements of a utility Grid and presents an abstract model to conceptualise essential infrastructure needed to support this vision. Then, a taxonomy and survey of the current market-oriented and system-oriented schedulers is provided, examining the contribution and the outstanding issues of each system in terms of utility Grid's requirements. This survey is intended to present the state-of-art and identify strength and weakness in the field [175][23].

2.1 Overview of Utility Grids and Preliminaries

A **utility Grid** imitates a market scenario consisting of the **two key players** (i.e. Grid Service Consumers (GSCs) and Grid Service Providers (GSPs)). Each of these players is generally self-interested and wants to maximise their utility (Figure 2.1). Consumers are users who have resource requirements to execute their applications. The resource requirement varies depending on the application model. For instance, parallel applications demand multiple CPUs at the same time with equal configuration, and similar network bandwidth between resources. The consumers are willing to compensate a providers for using their resources in the form of real money or barter [16]. On the other hand, providers are the owners of resources (i.e. disk, CPU) which satisfy consumer needs.

They can advertise their resources using other agents of the Grid such as Grid Market Directories [177]. It is the responsibility of resource providers to ensure user's application get executed according to Service Level Agreement (SLA) [42] signed with the consumer.

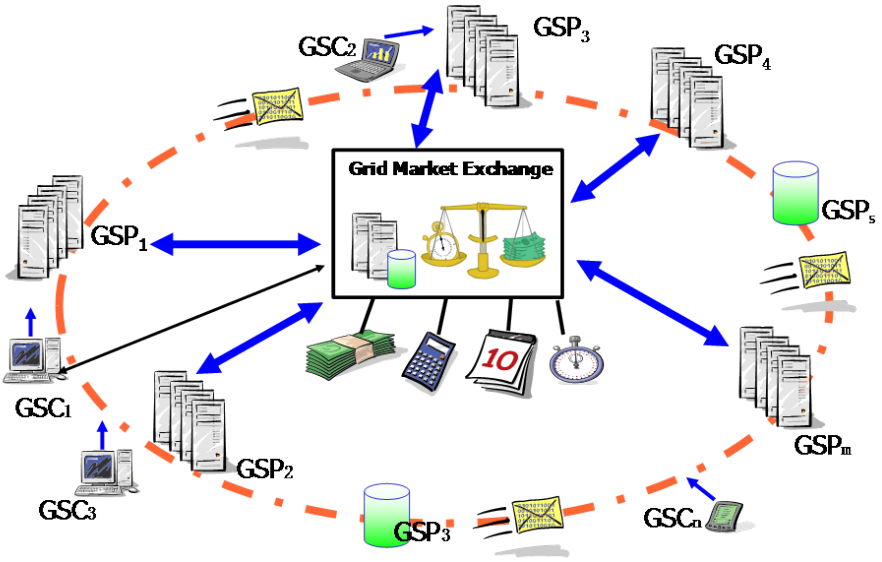


Figure 2.1: A Grid Market Exchange Managing Self-Interested Entities (Providers and Consumers)

To ease and control the buying and selling process there are other players in the utility Grid, such as **Grid market place** or **Grid market exchange** [121][127], which allow various consumers and providers to publish their requirements and goods (compute power or storage) respectively. The market exchange service provides a shared trading infrastructure designed to support different market-oriented systems. It provides transparent message routing among participants, authenticated messages and logging of messages for auditing. It can coordinate the users and lower the delay in acquiring resources. Moreover, it can help in price control and reduces the chances of the market being monopolised. **Personalised Brokers** are another kind of middle agents which, based on users' QoS requirements, peform the function of resource monitoring, resource discovery, and job submission. These brokers hide all the complexity of Grids from users. Each of the utility Grid players (i.e. consumer (or user agents such as personalised broker), provider, market exchange) has different requirements and goals. These requirements are discussed in detail, and summarised in the next section.

2.2 Requirements

In this section, we discuss the main bottlenecks or infrastructural enhancements required for utility Grids. In general, consumers and providers need mechanisms and tools that

facilitate the description of their requirements and decision making to achieve their goals, such as minimisation of monetary cost while meeting QoS requirements.

2.2.1 Consumer Requirements

User-centric Brokers: These brokers are user agents which discover and schedule jobs on to resources according to the user priorities and application's QoS requirements such as budget, deadline, and number of CPU required [163]. These brokers hide the heterogeneity and complexity of resources available in the Grid. On behalf of users, user-centric brokers provide functionalities such as application description, submission and scheduling; resource discovery and matching; and job monitoring. The user broker can also perform negotiation and bidding in an auction conducted by a market exchange or provider for acquiring resources.

Bidding/Valuation Mechanism: In the utility Grid, a variety of market models can exist simultaneously such as a commodity and auction market. To participate in both of the market model, users need to know the valuation of their applications in the form of budget which estimates the user's requirements. For example, in an auction market, users need to bid in order to grab a resource, and in such cases, budget or valuation can help brokers to bid on behalf of the users. In summary, consumers need utility models to allow them to specify resource requirements and constraints.

Market-Oriented Scheduling Mechanisms: In traditional Grids, generally users want to schedule their applications on the resources which can provide the minimum response time, and satisfy other QoS requirements in terms of memory and bandwidth. In utility Grids, one additional factor comes into the picture i.e. monetary cost, which requires new mechanisms as a user may relax some of her other requirements to save on monetary cost. Thus, one of the objectives of new scheduling mechanisms will be to execute the user application on the cheapest resource which can satisfy the user's QoS requirements. The market-oriented mechanisms can vary depending on the market model, user's objective (such as reduce time or cost), and application model (require multiple types of resources).

Allocation of Multiple Resources: Depending on the application model, consumers may want to run their applications on multiple resources, provided by more than one resource provider. For example, scheduling of a large parameter sweep across a number of providers, performing distributed queries across multiple databases, or creating a distributed multi-site workflow. Thus, brokers should have capabilities to schedule applications and obtain resources from multiple resource sites.

Estimation of Resource Usage: In general, due to the heterogeneity of hardware and different input sizes, it is difficult to describe precisely execution time, and the requirement of an application which can vary drastically. In the traditional Grid, an important research problem is how to profile the runtime of an application since execution time can affect not only the resource utilisation but also cause delays for users. In the utility Grid, this requirement becomes more critical as over estimation and under estimation of resource requirements can lead to tangible loss in the form of real money. Currently, the resource providers such as Amazon sell their compute resources in time blocks. In addition, if many users compete for the same resource, the resource availability, depending on individual user's requirement, can vary from minutes to days. Thus, users must estimate their resource needs in advance. Thus, profiling tools and mechanisms are required for efficient resource allocation in terms of utility.

2.2.2 Resource Provider Requirements

Resource Management Systems (RMSs): These systems interact with underline hardware infrastructure and control the allocation of resources and job scheduling. In a market-oriented system, the advance reservation function is required to allocate resources in future, and also to track the availability of resources which can be advertised. Thus, the reservation system should be able to provide the guaranteed resource usage time (based on SLA) and support estimation of the future resource offers. Grid Middleware such as Globus has components such as advance reservation, but to support the market-oriented reservation and scheduling, they should be integrated with a module that supports various market-oriented scheduling mechanisms and models.

Pricing/Valuation Mechanism: In utility Grids, a resource provider's main objective is to maximise their profit not just the efficiency of the system, thus mechanisms are required to set the resource price based on the market supply and demand, and the current level of resource utilisation. These prices can be static or can vary dynamically based on the resource demand. For example, an academic user may require more resources due to a conference deadline and, thus, is willing to pay more.

Admission Control and Negotiation Protocols: As stated before, in market-oriented systems, all participants are self-interested and want to maximise their utility. Thus, providers need to decide which user application they should accept or negotiate for. Since there may be a chance of reservation cancellation by users, thus the mechanisms such as resource over provisioning may be required. SLA is also needed to be formulated once a user request is accepted for reservation. In addition, depending on how providers want to

lease their resources, they may choose different market model for negotiation. For example, the simplest negotiation is required in the commodity models, while the bargaining model requires more intelligent negotiation protocol.

Commoditization of the Resources: Unlike many other markets, commoditization of the resources is one of the major difficult problems, which complicates the reservation and allocation decisions. For instance, for a compute intensive application, it is meaningless to just allocate CPUs without some memory. How much memory should be allocated when hardware infrastructure contain shared memory? Even for storage some small CPU cycles are required. Thus, intelligent resource partitioning techniques are required to overcome the hardware difficulties.

2.2.3 Market Exchange (ME) Requirements

An Information and Market Directory is required for advertising participants, available resources, and auctions. It should support heterogeneous resources, as well as provide support for different resource specifications. This means that the market ought to offer functionalities for providing, for instance, both storage and computation with different qualities and sizes.

Support for Different Market Models: Multiple market models are needed to be designed and deployed as resource providers and consumers have different goals, objectives, strategies, and requirements that vary with time [121]. If there are multiple sellers for the same good, a Double auction which aggregates supply and demand generally yields higher efficiency. If there is only one seller (e.g. in a differentiated service market for complex services), supporting single-sided auction protocols may be desirable. The negotiation protocol also depends on the user application. For example, in the case applications with soft deadlines, the large scheduling cycle helps in collecting a higher number of bids and offers for auction. This may lead to more efficient allocation than clearing continuously, since the allocation can be based on more resource information and has more degrees of freedom in optimising efficiency (and/or other objectives). On the contrary, users having urgent resource requirements may prefer an immediate allocation, thus the commodity model will be better choice for negotiation. Consequently, the market exchange must clearly support multiple negotiation protocols.

Reputation and Monitoring System: In general, it is assumed that after the scheduling mechanism has determined the allocation and resultant pricing, the market participants adhere to the market's decisions and promises. In reality, however, this does not happen due

to several reasons such as untruthful behaviour of participants, failure while communicating the decision, and failure of resources. Consequently, there is a need for reputation mechanisms that prevent such circumstances by removing distrustful participants. However, there is a strong need for monitoring systems that can detect any SLA violation during execution. In Grids, the reason for a job failure or a corruption of results is hard to detect since it can occur due to several reasons such as intentional misbehaviour of the resource provider, or technical reasons which are neither controlled by the user nor the provider. The monitoring systems should support reputation system for early detection of violations and responsible participant. An important challenge is thus to design such intelligent monitoring systems.

Banking System (Accounting, Billing, Payment Mechanism): In the market exchange, an accounting system is necessary to record all the transactions between resource providers and consumers. The accounting system especially records the resource usage and charges the consumer as per the usage agreement between the consumer and provider.

Meta-Scheduling/Meta-Brokering: The market exchange provides the services such as meta-scheduling of consumer applications on multiple resource providers in the case several consumers require simultaneous access to resources. For instance, a large parameter sweep application requires resources across the number of providers, performing distributed queries across multiple databases, or creating a distributed multi-site workflow. Thus, the meta-scheduling service does two tasks for their clients (i.e. resource discovery and efficiently scheduling applications according to client's objectives). It can act as an auctioneer in case a client wants to hold an auction.

Currency Management: For ensuring the fair and efficient sharing of resources, and a successful market, a well-defined currency system is essential. Two kinds of currency models are proposed i.e. virtual and real currency. Both of these currency models have advantages and disadvantages based on their managerial requirements. The virtual currency is generally deployed [146] due to its low risk and low stakes in case of mismanagement or abuse. However, virtual currency requires careful initial and ongoing management, and lacks flexibility. For buying and selling resources in a real commercial environment, the real currency is preferred due to several reasons. The most important reason is that the real currency formats (e.g. USD, Euro, etc.) are universally recognised and are easily transferable and exchanged, and are managed outside the scope of a Grid market exchange, by linked free markets and respective government policy.

Security and Legal System: To avoid spamming, there should be a security system for user registration. All the services of the exchange must be accessed by authorised users.

Similarly, there is also a need for legal support that can resolve various conflicts between providers and consumers, such as violations of SLA [42]. Thus, the legal support can be build by the help of some authoritative agency such as a country's government.

2.3 Utility Grid Infrastructural Components

Based on above requirements, in this section we discuss various infrastructure required for a fully functional utility Grid. Figure 2.2 outlines an abstract model for utility Grid that identifies essential components. This model can be used to explore how existing Grid middleware such as user-centric brokers, meta-schedulers and RMSs can be leveraged and extended to incorporate market-oriented mechanisms.

The utility Grid consists of multi-layer middleware for each participant: users (Grid application, user level middleware), Grid exchange, and providers (core middleware and Grid fabric). The architecture of each of the components should be generic enough to accommodate different negotiation models for resource trading. Except for Grid exchange and highlighted components, most of the components are also present in traditional Grids.

The lowest layer is the Grid fabric that consists of distributed resources such as computers, networks, storage devices, and scientific instruments. These computational resources are leased by providers, thus the resource usage is needed to be monitored periodically to inform the above layers about free resources which can be rented out. The resource managers in this layer have the responsibility of scheduling applications.

The core middleware offers the interface for negotiation with Grid exchange and user-level middleware. It offers services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of QoS such as resource reservation and trading. These services hide the heterogeneity at the fabric level. The support for accounting, market model and pricing mechanisms is vital for resource providers since such support enables them to participate in the utility Grid. The pricing mechanism decides how resource requests are charged. The pricing of resource usage by consumers can depend on several variables such as submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand). Pricing serves in the utility Grid as an efficient and cost-effective medium for resource sharing. The accounting mechanism maintains the actual usage of resources so that the final cost can be computed and charged from the consumers. The market model defines the negotiation protocol that can be used to serve different resource requests depending on their effect on provider's utility.

The user-level Grid middleware and applications also need to be enhanced to satisfy the requirements discussed in the previous section. A new layer is needed in the brokers to give users functionality of automatic bidding and negotiation. This layer also discovers

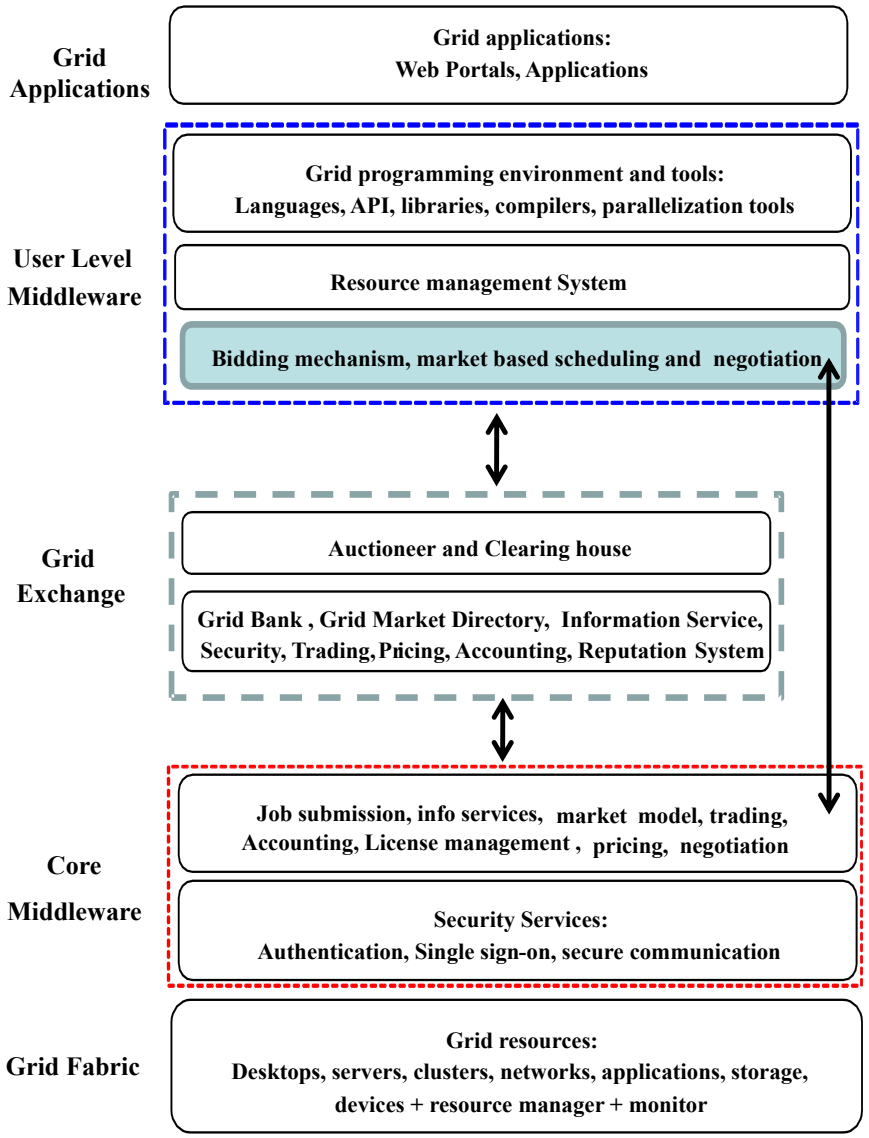


Figure 2.2: Utility Grid Component

resources based on a user's requirements such as deadline and budget. Automated negotiation capabilities are needed to be added to allow brokers to interact with Grid exchange and provider's middleware, and maintain SLAs.

In traditional Grids, the users and providers generally interact on a one-to-one basis rather than using third party services. Similar to other markets, in utility Grids, since negotiation with a provider is more complex due to the involvement of money and flexible resource reservation, the third party services become essential. Thus, utility Grids require Grid exchange middleware which can act as a buyer or seller on behalf of users and resource providers respectively. They require the capability of auctions and clearing house to match user resource demand to available resources. This middleware needs infrastruc-

tures such as a meta-broker which matches of users and providers, Grid Market Directory (GMD) that allows resource advertisements, negotiation protocols, a reputation system, security and price control. Meta-broker is a key component of the market-exchange. It acts as an auctioneer or clearing house, and thus schedules a user's application on the desired resources. More details on the meta-broker architecture used in this thesis are discussed in the next chapter.

2.4 Taxonomy of Market-Oriented Scheduling

There are several proposed taxonomies for scheduling in distributed and heterogeneous computing [175][5]. However, none of these taxonomies focus on the market-oriented scheduling mechanism in Grids. Here, we present the taxonomy, which emphasises the practical aspects of market-oriented scheduling in Grids and its difficulties which are vital to achieve utility-based Grid computing in practice. We can understand works in market-oriented scheduling (as shown in Figure 2.3) from five major perspectives, namely *market model, allocation decision, market objective, application model,* and *participant focus*.

2.4.1 Market Model

The market model refers to the mechanism used for trading between consumers and providers. Any particular market model cannot solve all the special requirements of different participants. Having different characteristics each model is the most profitable to its participants depending on the Grid situation. For example, when the number of participants in terms of consumers and providers is almost same, the Double auction is a much better choice. Due to the differences in the applicability of auctions, various authors have applied and analysed the efficiency achieved [121]. Various market models that can be applied to market-oriented Grid computing, include the following:

Game Theory

If Grid participants only interact in the form of an allocation game with different payoffs as a result of specific actions employing various strategies, a game theoretical setting can be assumed. This approach is generally applied to ease the congestion of a common resource or network which can lead to reduction in the overall utility of the system. There are two types of solution approaches in this context:

1. To avoid excessive consumption of the resources one can use a game with self-interested economic agents (non-cooperative games).

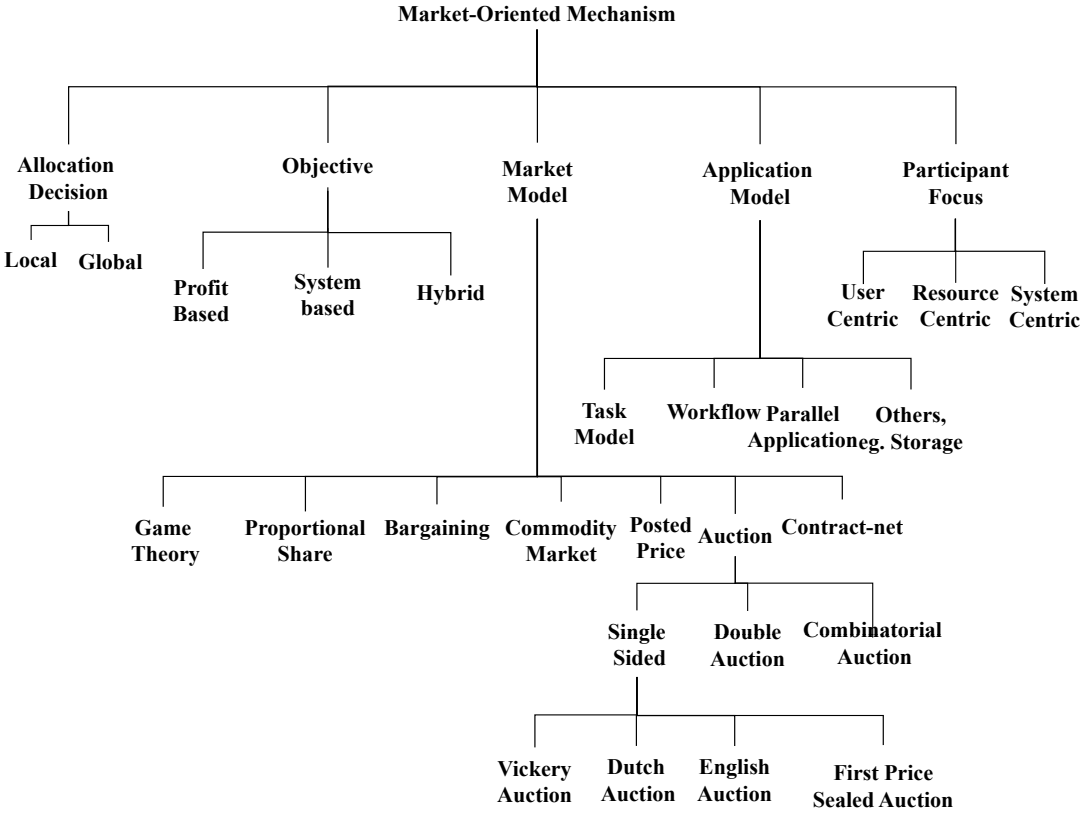


Figure 2.3: Taxonomy of Market-based Mechanisms

2. To achieve a load balancing effect, the unselfish distribution of tasks over resources can be achieved by using cooperative game agents.

The cooperative and non-cooperative games for resource sharing and allocation often employ a "Nash bargaining" approach, where bargainers negotiate for a fair "contract point" within feasible solution set [Nash 1950]. The application of game theory-based approaches is not very common for resource allocation in Grids. Feldman et al. [59] indicated in their analysis that the price-anticipating allocation scheme can result in higher efficiency and fairness at equilibrium. In their approach, the resource price is estimated by total bids placed on a machine. Kwok et al. [103] proposed a Nash equilibrium-oriented allocation system with hierarchical organised game. They used a reputation index instead of virtual budget or monetary unit for job acceptance decisions.

Proportional Share

The proportional share introduced and implemented in real cluster based systems such as Tycoon [104]is to decrease the response time of jobs and to allocate them fairly. Neumann et al. [153] proposed a similar approach such as proportional share where shares

of resources are distributed using a discriminatory pay-as-bid mechanism to increase the efficiency of allocation and for the maximisation of resource provider profit. This model makes an inherit assumption that resources are divisible, which is generally not the case when a single CPU is needed to be allocated; a situation which is quite usual in cooperative problem-solving environments such as clusters (in single administrative domain).

In the proportional share based market model, the percentage of resource share allocated to an application is proportional to the bid value in comparison to other users' bids. The users are allocated credits or tokens, which can be used for gaining access to resources. The value of each credit depends on the resource demand and the value that other users place on the resource at the time of usage. One major drawback of proportional share is that users do not get any QoS guarantee.

Commodity Market Model

In this model, resource providers specify their resource price and charge users according to the amount of resources they consume. The resource allocation mechanisms consist of finding prices and allocations such that each economic participant maximises their utility. One of the first evaluation work in Grids on commodity market was presented by Wolski et al. [168] who analysed and compared commodity markets with other auction models. Many commercial providers such as Amazon [6] are using commodity market models with fixed and dynamic pricing.

The determination of the equilibrium price is very crucial since it act as a great tool in resource allocation decisions. The prices depend on various factors such as investment and management cost of resource provider, current demand and supply, and future markets [148][3]. According to the prices, users can adopt any strategy to decrease their spending while getting satisfactory QoS level [140][151]. Various systems have been designed to automate this process; such as Dornemann et al. [47] designed and implemented of a workflow system based on a Business Process Execution Language (BPEL) to support on-demand resource provisioning. Ernemann et al. [52] presented an economic scheduling system for Grid environments. HA-JES (Kang et al. [91]) presented an algorithm to increase revenue for providers whose resources are underutilised, by ensuring high availability to users. To determine the equilibrium price, Stuer et al. [154] presented a strategy for commodity resource pricing in dynamic computational Grids. They proposed some refinements to the application of Smale's method for finding price equilibria in such a Grid market for price stability, allocative efficiency, and fairness.

Contract-Net

In the contract-net protocol, users advertise their demand and invite resource owners to submit bids [149][171] [90]. Resource owners check these advertisements with respect to

their requirements. If the advertisement is favourable, the resource owners respond with bids. The user consolidates all the bids, compares them, and selects the most favourable bids. The bidding process has only two outcomes: the bid is accepted or rejected in its entirety.

Bargaining

Bargaining models are employed in bi-lateral negotiations between providers and consumers, and do not rely on 3rd parties to mediate the negotiation. During the negotiation, each player applies concessions until mutual agreement is reached by alternating offers [162]. Li and Yahyapour [107] proposed a concurrent bilateral negotiation model for Grid resource management. The bargaining problem in Grid resource management is difficult because while attempting to optimise utility, negotiation agents need to: (i) negotiate for simultaneous access to multiple resources, (ii) consider the (market) dynamics of a computational Grid, and (iii) be highly successful in acquiring resources to reduce delay overhead in waiting for resources.

Posted Price

It is similar to the commodity market model. In this model, providers may also make special offers such as discounts for new clients; differentiate prices across peak and off-peak hours. Prices do not vary relative to the current supply and demand but are fixed over a period of time.

Auction

An Auction is a process of trading resources by offering them up for bidding, and selling the items to the highest bidder. In economic terms, it is also a method to determine the value of a resource whose price is unknown. An auction is organised by an auctioneer, who distributes Grid resources from the providers to the users. Thus, the mechanism consists of determining the winner and setting the price. The auctions can be divided into three types based on participants and commodity exchanged: a) Single-sided auction, b) Double-sided auction, and c) Combinatorial auctions.

- **Single-sided Auction**

  Single-sided auctions are mechanisms, where only buyers or sellers can submit bids or asks. Even though the single-sided auction is the most widely applied market model, it often leads to inefficient allocation [143]. The most prominent single sided auctions are Vickrey Auction, Dutch Auction, First Price Sealed Bid (FPSB), and English Auction.

1. **English Auction**

   In the English auction, the auctioneer begins the auction with a reserve price (lowest acceptable price) [43]. Auction continues in rounds with increasing bid prices, until there is no price increase. The item is then sold to the highest bidder.

2. **Dutch Auction**

   In the Dutch auction the auctioneer begins with a high asking price which is lowered until some participant is willing to accept the auctioneer's price or a predetermined minimum price is reached. That participant pays the last announced price. This type of auction is convenient when it is important to auction resources quickly, since a sale never requires more than one bid.

3. **Vickrey Auction**

   A Vickrey auction is a sealed-bid auction, where bidders submit sealed bids. The highest bidder wins, paying the price of the second highest bid. This gives bidders incentives to bid their true value. When multiple identical units are auctioned, one obvious generalisation is to have all bidders pay the amount of the highest non-winning bid.

4. **First Price Sealed Bid (FPSB) Auction**

   In this type of auction, all bidders simultaneously submit bids so that no bidder knows the bid of any other participant [43]. The highest bidder pays the price they submitted. In this case, the bid strategy is a function of one's private value and the prior belief of other bidders' valuations. The best strategy is to bid less than its true valuation and it might still win the bid, but it all depends on what the others bid.

- **Double-Sided Auction**

  In a Double auction, both providers and users submit bids which are then ranked (from the highest to the lowest) to generate demand and supply profiles. From the profiles, the maximum quantity exchanged can be determined by matching selling offers (starting with the lowest price and moving up) with demand bids (starting with the highest price and moving down). This format allows users to make offers and providers to accept those offers at any particular moment. In the Double auction, the winner determination depends on different aspects such as aggregation, resource divisibility, and goods which can be homogeneous or heterogeneous. Aggregation can come from the supplier side or from the buyer side. If no aggregation is allowed then each bid can be exactly matched to one ask. Divisible goods can be allocated partially. In the case that the bidder wants the entire good or nothing

then its bid is considered indivisible. Kant et al. [92] proposed and compared various types of Double auctions to investigate its efficiency for resource allocation in Grid. In addition, Tan et al. [156] proposed the stable continuous Double auction to overcome the problem of high volatility.

- **Combinatorial Auctions**

  The Grid users may require a combination of multiple resources such as CPUs, memory, and bandwidth. Combinatorial auction allows users and providers to trade a bundle of multiple resources. It is advantageous to users as they do not need to participate in multiple negotiations with providers for each resource required. Moreover, in some cases it also leads to cost benefits. In combinatorial auction, users express their preferences as bundles of resources that need to be matched. The providers submit their asks and the auctioneer solves the optimisation problem of allocation. The only drawback of combinatorial auction is the NP-hardness [118] of the matching problem, which makes it inapplicable for large scale settings. A number of variants of combinatorial auction are proposed in the literature to allocate computational resources among Grid users [94][142].

2.4.2 Allocation Decision

In Grids, the resource allocation to users can be done at two points. It can be initiated either by an individual provider (local) or a middleman such as meta-scheduler or auctioneer (global). In 'local', the trading decisions are based on the information provided by one resource provider. Generally in this case, users approach the resource provider directly to buy or bid for a resource bundle advertised. For instance, to buy compute resources of Amazon [6], users can directly negotiate with Amazon service [6]. Most of the single-sided auctions fall into this category.

In 'global', the trading decisions are based on the global information of multiple providers. Users utilise the services of a an auctioneer in the market exchange to get the required resources. Thus, the auctioneer makes the decision on behalf of the users to buy resources from providers. The two-sided auctions fall into this category.

The local decision point is more scalable but can lead to contention. While the global decision point is more optimised and coordinates demand fairly.

2.4.3 Participant Focus

The two major parties in Grid computing, namely, resource consumers who submit various applications, and resources providers who share their resources, usually have different

motivations when they join the Grid. The participant focus identifies the market side for which market oriented systems or mechanisms are explicitly designed to achieve benefit.

Application-centric: In the application centric, mechanisms are designed such that an application can be executed on resources, which meet the user requirements within his/her budget or deadline.

Resource-centric: Similarly, a resource-centric mechanism focuses mainly on resource providers by fulfilling their desired utility goal in terms of resource utilisation and profit.

System-centric: In the market scenario, there may be middlemen such as meta-brokers or meta-schedulers that coordinate and negotiate the resource allocations between users and producers. They try to maximise the utility for both users and providers. Thus, the resource allocation decision involves multiple users and providers.

In utility Grids, mechanisms are required that can cater to the needs of both sides of the market. For instance, they should be able to satisfy end-users' demand for resources while giving enough incentive to resource providers to join the market. Moreover, the specific requirements of participants should not be neglected. It is also possible for market-oriented RMSs to have multiple participants focus such as in Double auctions.

2.4.4 Application Type

Market-oriented resource allocation mechanisms need to take into account job attributes to ensure that different job types with distinct requirements can be fulfilled successfully. The application model affects not only the scheduling mechanism but also other aspect of the utility Grid such as resource offerings by providers, negotiation with providers, and formation of SLAs and their monitoring. For applications consisting of independent tasks, all the tasks can be distributed across multiple providers and thus optimisation of the user's utility is easier. For the parallel application model, all tasks are needed to be mapped on a single resource site. In the workflow type of application, there is a precedence order existing in tasks; that is, a task cannot start until all its parent tasks are done. This will require coordination between multiple providers, and the scheduling problem needs to be fault tolerant since single failure may result in a large utility loss.

2.4.5 Allocation Objective

The market-oriented mechanisms can be used to achieve different objectives both in utility Grids and traditional Grids. The allocation objective of mechanism can be profit-oriented,

system-oriented, or hybrid of both of them. The objectives of various participants decide the trading relationship between them. The profit based objective in terms of monetary gains, in general, encourages competition between participants. Thus, each participant tries to maximise their own utility without considering other consumers.

The objective of market-oriented scheduling mechanisms could be to achieve optimisation of system metrics such as utilisation, fairness, load balancing, and response time. This application of market-oriented mechanism, categorised in taxonomy as "system-oriented objective", is quite common. For example, OurGrid [7] uses a resource exchange mechanism termed network of favours which is used to share resources among distributed users. Bellagio [9] is another system deployed on PlanetLab for increasing system utilisation on non peak time. The objective of market-oriented scheduling mechanism can be of hybrid type. For example, a user may simultaneously want to minimise the response time and the cost of application execution. A provider may accept a less profitable application to increase resource utilisation rather than waiting for more profitable jobs.

2.5 Survey of Grid Resource Management Systems

Grid RMSs chosen for the survey can be classified into two broad categories: market-oriented or system-oriented (presented in the following sections). Since this survey focuses on utility computing, market-oriented RMSs are surveyed to understand current technological advances and identify outstanding issues that are yet to be explored so that more practical RMSs can be implemented in future. On the other hand, surveying system-oriented RMSs allow analysing and examining the applicability, and suitability of these systems for supporting utility Grids in practice. This in turn helps us to identify possible strengths of these systems that may be leveraged for utility computing environments. In traditional Grids, a user accesses the Grid services either through User Brokers (UB) or Local Resource Managers (LRM). These systems schedule the jobs using system-centric approaches which optimise the metrics such as response time and utilisation. We call such schedulers as "System-Oriented Schedulers" to differentiate them from the schedulers in market-oriented Grids (or utility Grids).

2.5.1 Survey of Market-Oriented Systems

Table 2.1 shows a summary listing of the existing market-oriented RMSs and brokers that have been proposed by researchers for various computing platforms. In the market-oriented Grid, there are also three entities which participate in scheduling. One is the user broker that provides access to users on multiple resources. The resource providers are other entities who also have resource brokers that facilitate the admission control,

pricing, and negotiation with user brokers. The negotiation between users and resource providers can be on one-to-one basis or through third entity i.e. market exchange. The market exchange also provide other services such as resource discovery, banking, buying and selling compute services.

Gridbus Broker

Gridbus broker [163] is a single user resource broker that supports access to both computational and data Grids. It transparently interacts with multiple types of computational resources which are exposed by various local Grid middleware's such as Globus, Alchemi, Unicore and Amazon EC2, and scheduling systems such as PBS and Condor. By default, it implements two scheduling strategies that take into account budget and deadline of applications. Additionally, the design of the broker allows the integration of custom scheduling algorithms. Job-monitoring and status-reporting features are provided. Gridbus broker supports two application models, i.e., parametric-sweep and workflow.

Nimrod/G

Nimrod/G is an automated and specialised resource management system, which allows execution of parameter sweep applications on Grid to scientists and other types of users. Nimrod/G follows mainly the commodity market model and provides four budget and deadline based algorithms [25] for computationally-intensive applications. Each resource provider is compensated for sharing their resources by the users. The users can vary their QoS requirement based on their urgency and execution expense. Nimrod/G consists of a Task Farming Engine (TFE) for managing an execution, a Scheduler that talks to various information services and decides resource allocations, and a Dispatcher that creates Agents and sends them to remote nodes for execution. It is widely used by scientific community for their computation-intensive simulations in the areas of bio-informatics, operations research, ecological modelling, and Business Process Simulation.

Tycoon

Tycoon [104] is a market-based distributed resource allocation system based on Proportional Share scheduling algorithm. The user request with the highest bid is allocated the processor time slice. The bid is computed as the pricing rate that the user pays for the required processor time. Tycoon allocates the resources to the self-interested users in environments where service hosts are unreliable because of frequent changes in the availability. Tycoon distinguishes itself from other systems by separating the allocation mechanism (which provides incentives) from the agent strategy (which interprets preferences). This simplifies the system and allows specialisation of agent strategies for dif-

ferent applications while providing incentives for applications to use resources efficiently and resource providers to provide valuable resources. A host self-manages its local selection of applications, thus maintaining decentralised resource management. Hosts are heterogeneous since they are installed in various administrative domains and owned by different owners. Tycoon's distributed market allows the system to be fault tolerant and to allocate resources with low latency.

Spawn

Spawn [95] uses sealed-bid second-price auctions for market-oriented resource allocation in a network of heterogeneous computer nodes. Users place bids to purchase CPU resources for executing hierarchy-based concurrent programs in auctions held privately by each computer node and are not aware of other users' bids. The concurrent applications are then represented using a tree structure where a hierarchy of tasks expand or shrink in size depending on the resource cost. This mechanism limits the ability of customers to express fine-grained preferences for services.

Java Market

Java Market [173] is one of the oldest market-oriented systems developed by John Hopkins Univ. It is an Internet-wide meta-computing system that brings together people who have applications to execute and people who have spare computing resources. One can sell CPU cycles by pointing a Java-enabled browser to a portal and allow execution of Applets in a QoS-based computational market. The goal of Java Market is to make it possible to transfer jobs to any participating machine. In addition, in Java Market, a resource provider receives payment or award based on the execution time of a job and the amount of work done.

Mariposa

Mariposa [152] is a distributed database system developed at the University of California. It supports query processing and storage management based on budget. Users submit queries with time-dependent budgets to brokers who then select servers for executing the queries based on two protocols. One protocol is expensive as it solicits bids from all servers, requiring many communication messages. The expensive protocol adopts a greedy algorithm that aims to minimise cost to schedule sub-queries so as to select the cheapest server for the user. The other protocol is cheaper since it selects specific server based on the historical information. In Mariposa, bids on queries are based on local and selfish optimisation of each user.

NAME	Architecture	Economic Model	Mechanism	Traded Commodity	Pricing	Target Platform	Application Model	User Role
Gridbus (UB)	Not Applicable (NA)	Commodity Market	Time and Budget based Algorithm	Compute and Storage	NA	Commercial Providers	Bag-of-Task and WorkFlow	Budget and Time
Nimrod/G (UB)	Centralised	Commodity Market, Spot Market, and Contract-Net for price establishment	Deadline and Budget Constrained Algorithms	CPU Cycles and Storage	Fixed pricing	World Wide Grid (resources Grid enabled using Globus middleware)	Independent multiple tasks and data parallel applications	Time and Budget
Tycoon (RMS)	Distributed, Centralised	Proportional Share	Proportional Share	CPU cycles	Pricing based on bids	Clusters, Grids	Task Allocation	Discrete bid
Spawn (RMS)	Decentralised	Auction	Vickery Auction	CPU time	Second price	Cluster	Task allocation	Discrete bid
Java Market (ME)	Centralised	Commodity Market	Cost based Greedy	Compute	Fixed	WWW	Java program	Bidding done by resources
Mariposa (RMS)	Centralised	Commodity Market	Cost minimisation	Storage	Pricing based on load and historical info.	Distributed database	Data	Budget and queries
GRIA (RMS)	P2P	Contract-based	Not Specified	Compute	Through negotiation between providers	Grid	Not Specified	NA
PeerMart (RMS)	P2P	Auction	Double Auction	Not Specified	Mean Price based on matched ask and bid	P2P	Not Specified	Bids
Bellagio (RMS)	Centralised	Combinatorial Auction	Vickery Auction	CPUs and Storage	Second price	P2P	Not Specified	Bidding

Sharp (ME)	Centralised	Commodity Market	Leases	CPU time	Fixed	Grid	Lease allocation	Lease request to Sharp
Shirako (ME)	Centralised	Not specified seems to be commodity	Negotiation, leasing generic	Virtual Machine and Storage	NA	Virtual Machines	Not specified	Lease request to broker
OCEAN (ME)	Distributed	Bargaining, Tendering, Contract-Net, Continuous Double Auction	Discovering potential seller and Negotiation	CPU Cycles	Fixed	Any distributed resource	Not Specified	Discover and Negotiation.
CatNets (ME)	Decentralised	Bargaining	Negotiation	Complex Services and Basic Services	Through negotiation. Dynamic pricing depend on available servers	Grid and Service-Oriented Computing	Not Specified	Bidding type
SORMA (ME)	Centralised	Combinatorial Auction	Greedy Mechanism	Not Specified	K-Pricing, based on auction	Commercial Providers	Not Specified (Simulation are based on independent tasks)	Bidding
GridEcon (ME)	Centralised	Commodity Market	Not Specified	Resources managed by commercial service providers	Fixed	Commercial Resource Providers	Not Specified	Price specified by resource
G-Commerce (ME)	Centralised	Commodity Market	Auction	NA	Dynamic pricing	Simulates hypothetical consumers and produces	NA	Bidding

Table 2.1: Market-Oriented Scheduling Systems

GRIA

GRIA (Grid Resources for Industrial Applications) [155] is a web-services based Grid middleware for Business-to-Business (B2B) service procurement and operation. It aims at the development of business models and processes that make it feasible and cost-effective to offer and use computational services securely in an open Grid market exchange. It also helps the Industries to achieve better utilisation and manage demand peaks on resources. GRIA software is based on and uses web services standard specifications and tools such as Apache AXIS. GRIA aiming to make Grid Middleware reliable for industrial application, thus, provides various software packages for performance estimation and QoS, workflow enforcement, cluster management, security and inter-operability semantics. Thus, each service provider using GRIA middleware has an account service and a resource allocation service, as well as services to store and transfer data files, and execute jobs which process these data files. Service provider's interaction is based on B2B model for accounting and QoS agreement.

PeerMart

PeerMart [78] is a Peer-to-Peer market based framework which allows completely decentralised trading of services between peers. It includes with capability of dynamic pricing and efficient price dissemination, and services discovery over a P2P network. Using PeerMart, peers can bid prices for services, which enable them to govern the desired service performance. PeerMart implements an economically efficient distributed Double auction mechanism where each peer being responsible for matching several services. PeerMart uses the overlay network infrastructure to map the services onto particular sets of peers following a fully distributed and redundant approach for high reliability and scalability to the number of participating peers. Its main limitation is the tight integration of auction model in the framework, making it inflexible with respect of the market model.

Bellagio

The Bellagio [9] is a resource management system that allocates resources using combinatorial auction in order maximise aggregate end-user utility. Users identify their resources of interest via a SWORD [125] based resource discovery mechanism and register their preference to a centralised auctioneer for said resources over time and space as a combinatorial auction bids using a bidding language, which support XOR bids [122]. The bids are formulated in the virtual currency. The auction employed in Bellagio is periodic. Unlike other work that focuses on the contention for a single resource (CPU cycles), they are motivated by scenarios where users express interest in 'slices' of heterogeneous goods (e.g. disk space, memory, bandwidth). Bellagio employs Share [37] for resource

allocation in order to support a combinatorial auction for heterogeneous resources.

SHARP

SHARP [69] is not exactly a complete resource management system, but it is an architecture to enable secure distributed resource management, resource control and sharing across sites and trust domains. The real management and enforcement of allocations are created by resource provider middleware which process the tickets and leases issued by SHARP. SHARP stands for Secure Highly Available Resource Peering and is based around timed claims that expire after a specified period, following a classical lease model. The resource claims are split into two phases. In the first phase, a user agent obtains a 'ticket', representing a soft claim that represents a probabilistic claim on a specific resource for a period of time. In the second phase, the ticket must be converted into a concrete reservation by contracting the resources site authority and requesting a 'lease'. These two phases allow SHARP system to become oversubscribed by issuing more tickets than it can support. SHARP also presents a very strong security model to exchange claims between agents, either user agents or 3rd party brokers, and that then achieves identification, non-repudiation, encryption, and prevents man-in-the-middle attacks.

CATNET

CATNET Project [54] proposed a Catallaxy based market place where trading is divided into two layers, the application and service layer. The notion of a Catallaxy based market for Grids was proposed by Austrian economist F.A. von Hayek. In this market, prices evolve from the actions of economically self-interested participants which try to maximise their own gain whilst having limited information available to them. In the application layer, complex services are mapped to basic services. The service layer maps service requests to actual resources provided by local resource managers. There are two markets which operate simultaneously - one for buying resources by service providers from resource providers, and another for buying services by clients from service providers. Thus, the client is not aware of the details of the resource provider, and vice versa. The prices are fixed in two markets by bilateral bargaining. CATNETS offers very interesting features but lacks comprehensive support (e.g., monitoring, multi platform deployment).

In both layers, the participants have varying objectives which change dynamically and unpredictably over time. In the application/service layer, a complex service is a proxy who negotiates the access to bundles of basic service capabilities for execution on behalf of the application. Basic services provide an interface to access computational resources Agents representing the complex services, basic services and resources participate in a peer-to-peer trading network, on which requests are disseminated and when an appropriate provider is found, agents engage in a bilateral bargaining [55].

Shirako

Shirako [87] is a generic and extensible system that is motivated by SHARP for on-demand leasing of shared networked resources across clusters. Shirako framework consists of distributed brokers which provision the resources advertised by provider sites to the guest applications. Thus, it enables users to lease groups of resources from multiple providers over multiple physical sites through broker service. Site authorities compute and advertise the amount of free resource by issuing resource tickets to the selected brokers. When a broker approves a request, it issues a ticket that is redeemable for a lease at a site authority. The ticket specifies the resource type, resource units granted and the validation period. SHIRAKO allows 'flexible' resource allocation through leases which can be re-negotiated and extended via mutual agreement. A request can be defined as 'elastic' to specify a user will accept fewer resources if its full allocation is not available. Requests can be 'deferrable' if a user will accept a later start time than what is specified in the lease if that time is unavailable. The function of broker is to prioritise the request and match to appropriate resource type and quantity. Provider side is represented by site authorities that run Cluster on Demand [31] to configure the resources allocated at the remote sites.

OCEAN

OCEAN (Open Computation Exchange and Arbitration Network) [127] is a market based system for matching user applications with resources in the high performance computing environments, such as Cluster and Grid computing. It consists of all major components required to build utility Grid, such as user node which submit trade proposals, computational resource and underlying market mechanism. Ocean first discovers potential sellers by announcing a buyer's trade proposal using optimised P2P search protocol. Then, the user node can negotiate with sellers based on the rules dynamically defined in a XML format. The ability to define negotiation rules is a remarkable characteristic of OCEAN that allows the adaptation of the economic model to diverse applications. The two possible negotiation allowed by OCEAN are "yes/no" and automated bargain.

SORMA

Based on market engineering principles, the SORMA project [121] proposed an Open Grid Market which is built above the existing RMSs. It consists of self-interested resource brokers and user-agents. The users submit their bids for resources to the Open Grid Market using an autonomous bidding agent. On the other side of the market, the resource side bidding agents publish automatically available resources based on their predefined policies. The Open Grid Market matches requesting and offering bids and executes them against each other using Combinatorial Auction. The matches (i.e. allocations) are for-

mulated in SLAs (i.e. contracts). The Grid middleware is responsible for the resource provisioning and the payment system (such as PayPal) for the monetary transfer of funds. The open infrastructure of Open Grid Market allows various resource providers with different virtualisation platforms or managers to easily plug in the market.

GridEcon

GridEcon Project [4] proposed a market exchange technology that allows many (small and medium) providers to offer their resources for sale. To support buying and selling of resources, GridEcon market offers various services that makes exchange of commodity convenient, secure, and safe. The GridEcon market also proposed to design a series of value-added services on top of the market exchange (e.g. insurance against resource failures, capacity planning, resource quality assurance, stable price offering), ensuring quality of the traded goods for Grid users. Currently, GridEcon supports only commodity market model where commercial resource providers can advertise their spare resources. The fixed pricing is used to allow users to sell and buy resources. The GridEcon market delegates the real resource management to commercial service providers.

G-Commerce

G-Commerce [169] provides a framework for trading computer resources (CPU and hard disk) in commodity markets and Vickrey auctions. By comparing the results of both market strategies in terms of price stability, market equilibrium, consumer efficiency, and producer efficiency, the G-commerce project concludes that commodity market is a better choice for controlling Grid resources than the existing auction strategies. It is argued and shown in simulations that this model achieves better price predictability than auctions. Thus, G-commerce is a Grid resource allocation system based on the commodity market model where providers decide the selling price after considering long-term profit and past performance. The simulated auctions are winner-takes-it-all auctions and not proportional share, leading to reduced fairness and starvation. Furthermore, the auctions are only performed locally and separately on all hosts, leading to poor efficiency.

2.5.2 System-Oriented Schedulers

For over a decade various technologies have enabled applications to be deployed on the Grids, including Grid middleware such as Globus [64], Legion [30], and gLite [8]; schedulers such as Application Level Schedulers (AppLeS) [11]; and resource brokers including Gridbus Resource Broker [163], Nimrod/G [1], Condor-G [68], and GridWay [85]. These meta-schedulers or resource management systems interact with Local Schedulers or Grid

Middleware of various resource sites. The Local Scheduler supported such as Load Sharing Facility (LSF) [180], Open Portable Batch System (Open PBS [79] and Grid Engine (SGE / N1GE) [71]. In following section, we discuss some of the scheduling systems in detail and compare them using a Table 2.2.

Community Scheduler Framework (CSF)

The Community Scheduler Framework (CSF) [172] is an open source tool set for implementing a Grid meta-scheduler using the Globus Toolkit Services. The Grid meta-scheduler provides an environment for dispatching jobs to various resource managers. CSF was developed by Platform Computing in cooperation with the Jilin University, China. The CSF provides plug-in for various heterogeneous schedulers such as Platform Load Sharing Facility (LSF), Open Portable Batch System (Open PBS) and Grid Engine (SGE / N1GE), however CSF is designed for the best compliance with Platform LSF.

CSF implements by default two basic scheduling algorithms i.e. Round-robin and reservation based algorithm. For the later algorithm, CSF facilitates advance reservation of resources for its users. Thus, users can make resource reservation using Resource Manager Tool of CSF, in order to guarantee the resource availability at a specified time. In addition, it also provides submission and monitoring tools to its users.

Computing Centre Software (CCS)

CCS [132] is a vendor-independent resource management software that manages geographically distributed High Performance Computers. It is analogous to Globus and consists of three main components the CCS, which is a vendor-independent Local Resource Management Schedulers (LRMSs) for local HPC systems; the Resource and Service Description (RSD), used by the CCS to specify, and map hardware and software components of computing environments; and the Service Coordination Layer (SCL), which coordinates the resource usage across multiple computing sites. CCS schedules and maps interactive and parallel jobs using an enhanced first-come-first-served (FCFS) scheduler with backfilling. Deadline scheduling is another feature of CCS that gives the flexibility to improve the system utilisation by scheduling batch jobs at the earliest convenient and at the latest possible time. It also supports jobs with reservation requirements. At the meta-computing level, the Centre Resource Manager (CRM) is a layer above the CCS islands that exposes CSS scheduling features. When a user submits an application, the CRM maps the user request to the static and dynamic information regarding available resources through Centre Information Server (CIS). Centre Information Server (CIS) is a passive component that contains information about resources and their status. Once the CRM finds resources, it interacts with selected CCS islands for resource allocations. If not all resources are available, the CRM either re-schedules the request or rejects it.

GridWay

GridWay [85] is a meta-scheduler framework developed by a team working for Distributed Architecture Group from Universidad Complutense in Madrid, Spain. GridWay provides a transparent job execution management and resource brokering to the end user in a 'submit and forget' fashion. GridWay uses the Globus GRAM to interface with remote resources and, thus it can support all remote platforms and resource managers (for example fork, PBS and LSF) compatible with Globus. GridWay offers only simple scheduling capabilities even though custom scheduling algorithms are also supported. By default, GridWay follows the "greedy approach", implemented by the round-robin algorithm. The collective scheduling of many jobs is not supported by the meta-scheduler. GridWay also provides sophisticated resource discovery, scheduling, constant monitoring and self-adaptive job migration to increase performance. Thus, an application is able to decide about resource selection as it operates, i.e. it can modify its requirements and request a migration. GridWay also enables the deployment of virtual machines in a Globus Grid.

Moab (Silver) Grid Scheduler

Moab Grid Scheduler is a Grid metascheduler developed by Cluster Resources Inc. Moab allows combining the resources from multiple high performance computing systems while providing a common user interface to all resources. It supports intelligent load balancing and advanced allocation allowing a job to be run over multiple machines in a homogeneous way or in a heterogeneous way resulting in better overall utilisation and better time. Moab supports all major scheduling systems and even optionally relies on Globus Toolkit Grid middleware for security and user account management purposes. It manages the resources on any system where Moab Workload Manager (a part of Moab Cluster Suite) is installed. Moab Workload Manager is a policy engine that allows sites to control the allocation of available resources to jobs. The meta-scheduler supports fine-grained Grid level fairness policies. Using these policies, the system manager may configure complex throttling rules, fairshare, a hierarchical prioritisation, and cooperation with allocation managers. Moab also has support for advanced reservations. This feature enables scheduling techniques such as backfilling, deadline based scheduling, QoS support and Grid scheduling. One of the most interesting features going to be added in Moab is support for resource selection based on utility function where job completion time, resource cost, and other parameters are taken into account. These features allow easy transition of Moab meta-scheduler to market-oriented Grid.

Name	Allocation Mechanism	Scheduling Type	Scheduling Objective	Architecture	Application Type	Advance Reservation	Grid Middleware
CSF	Round-Robin and Reservation-based	Online	NA	Centralised	Task Model	Yes	LSF, Open PBS, SGE, Globus
CCS	FCFS	Online/ Interactive	Minimise Response Time	Decentralised	Parallel Application	Yes	NA
GridWay	Greedy/ adaptive scheduling	Online	Minimise Response Time	Centralised	Parallel application and Parametric Sweep	No	Globus
Maob (Silver)	Fairshare and Job prioritisation	Online	Load balancing and Response time minimisation	Centralised	Task Model	Yes	Globus, Maui Scheduler
Condor/G	Matchmaking	Online	NA	Centralised	Not Specified	No	Globus, Unicore, NorduGrid
Gruber/ Di-Gruber	FCFS	Online	Not Specified	Decentralised	Task Model	Yes	Globus
eNanos	Job Selection policies based on arrival time and deadline; Resource Selection based on EST	Batch/ Periodic	Minimise Response time	Centralised	Task Model	No	Globus
APST	Divisible Load Scheduling-based algorithms	Single application	Optimise Response time	Centralised	Parametric-Sweep	Yes/No	Globus

Table 2.2: System-Oriented Schedulers

Condor-G

Condor-G [68] is a fault tolerant job submission system that can access various computing resources which employs softwares from Globus and Condor [108] to allocate resources to users in multiple domains. Condor-G is not a real broker but a job manager, thus it does not support scheduling policies but it provides framework to implement scheduling architecture about it. Condor-G can cooperate with the following middleware: Globus Toolkit (2.4.x - 4.0.x), Unicore and NorduGrid, and it can submit jobs to Condor, PBS and Grid Engine (SGE / N1GE) scheduling systems. Condor's Classified Advertisement language (ClassAd) MatchMaking tool allows users to specify which resource to allocate. The mechanism allows both jobs and machines to describe attributes about themselves, their requirements and preferences, and matches result in a logical-to physical binding. The GlideIn mechanism is also provided in Condor-G that starts a daemon processes which can advertise resource availability which is used by Condor-G to match locally queued jobs to resources advertised. The command-line interface is provided to perform basic job management such as submitting a job, indicating executable input and output files and arguments, querying a job status or cancelling a job. The most striking capability of Condor-G is its failure management which can deal with crashes at various levels.

Gruber/DI-Gruber

To avoid bottleneck of a central broker, DI-Gruber [49] is implemented as a completely distributed resource broker. It has been developed as an extension of the SLA based GRUBER broker deployed on the Open Science Grid.

The Gruber system [48] consists of four main components. The engine implements several algorithms necessary for determining optimised resource assignments. The site monitor acts as a data provider that publishes the status of Grid resources. The site selector provides the information about sites which is used for selecting a resource provider for execution of new tasks. It communicates with the engine to select the resource provider. The queue manager resides on submitting hosts, deciding which jobs can be executed at what time. The Gruber can be utilised as the queue manager that controls the start time of jobs and enforces Virtual Organisation (VO) policies, or as a site recommender when the queue manager is not available.

eNanos Resource Broker

eNANOS [134] is a general purpose OGSI-compliant resource broker developed by the Barcelona Supercomputing Center. It abstracts Grid resources and provides an API-based model to access the Grid. The eNanos Grid resource broker is implemented on top of Globus Toolkit (GT) and supports both GT2 and GT3. It focuses on resource discov-

ery and management, failure handling, and dynamic policy management for job scheduling and resource selection. The eNanos Grid resource broker provides dynamic policy management and multi-criteria user requirements which are described in an XML document. These multi-criteria descriptions are used for resource filtering and ranking. The job scheduling in eNanos broker is divided into three phases, first is to select job to be schedule, second to select resource for selected job, and finally using meta-policy which consists of selection of the best job and the best resource. For job scheduling, several polices are implemented such as FIFOjobPolicy (First In First Out), REALTIMEjobPolicy (minimises *REALTIME=deadline time-estimated time of job finalisation*), EDFjobPolicy (Earliest Deadline First). Similarly for resource selection RANKresPolicy (resource selection based in the greatest rank obtained from the resource filtering process), ESTresPolicy (Earliest Starting Time, based in the estimated waiting time for a job in a local queue). Jobs are queued up in the local system, and periodically scheduled by the resource broker.

AppLeS Parameter Sweep Template (APST)

APST [11] is an application level scheduler that provides an environment for scheduling and deploying large-scale parameter sweep applications (PSAs) on Grid platforms. APST supports scheduling and job submission on different Grid middleware and schedulers that take into account PSAs with data requirements. The APST scheduler allocates resources based on several parameters including predictions of resource performance, expected network bandwidths and historical data. The scheduler takes help of tools such as DataManger and ComputeManager to deploy and monitor data transfers and computation respectively which in turn get information from sources such as Network Weather Service (NWS) [170] and the Globus Monitoring and Discovery Service (MDS) [95]. AppLeS interacts directly with resource managers, performs all application management tasks, including, e.g., file staging, and can enact collations of applications. APST is compatible with different low-level Grid middleware through the use of Actuators and also allows for different scheduling algorithms to be implemented.

2.6 Discussion and Gap analysis

After understanding the basic features of various market-oriented and system-oriented schedulers, based on presented taxonomy and requirements of utility Grids, we can identify several outstanding issues that are yet to be explored to adopt a Grid for creating a utility computing environment.

2.6.1 Scheduling Mechanisms

The market oriented scheduling mechanisms vary based on a market model used for trading resources. For example, if auction is the main market model then strategies for bidding and auction selection are required to maximise the chances of winning the auction. While in a commodity model, aggregation of resources from different provider is required to maximise user's utility. The challenges which are needed to be tackled more deeply can be categorised as following:

Support for multiple QoS parameters

In utility Grids, other than traditional QoS requirements of users such as response time, additional QoS issues are needed to be addressed. For example, for HPC application, one has to minimise the execution time, thus, the resource capability and availability becomes essential which may be contradictory to the budget constraint. Many factors, such as deadline, resource reliability and security, need to be considered with monetary cost while making a scheduling decision on utility Grids. Similarly, the resource management system should support QoS based scheduling and monitoring to deliver the quality service.

Support for different application type

The current market-oriented scheduling mechanisms mainly support simpler applications/ job models such as parameter-sweep. But, in reality, more advanced job models that comprise parallel job processing type and multiple-task data intensive applications, such as message-passing and workflow applications are also required by users. Thus, advanced algorithms, which require concurrent negotiation with multiple resource providers, are needed to be designed.

Support for market-oriented meta-scheduling mechanisms

Currently, most of the scheduling approaches in utility Grids are studied from an auction perspective [143] (Schnizler et al., 2008). However, auctions based scheduling may not be always suitable, particularly when users want immediate access to resources or they are part of the same community. As an example of a user community, we consider the financial institution Morgan Stanley that has various branches across the world. Each branch has computational needs and QoS (Quality of Service) constraints that can be satisfied by Grid resources. In this scenario, it is more appealing for the company to schedule various applications in a coordinated manner. Furthermore, another goal is to minimise the cost of using resources to all users across the community. Thus, mechanisms are required for user selection, and then resource allocation for utility maximisation across all users. To

tackle the problem of coordination and utility maximisation among concurrent users, this thesis proposes market-oriented meta-scheduling mechanisms in the three scenarios.

2.6.2 Market-Oriented Systems

In the previous sections, we discussed major systems which support market based mechanisms to buy and sell resources, and execute applications. Some of the most important outstanding issues from user, provider and market exchange perspective are presented as follows:

User Level Middleware

User level Middleware such as Gridbus broker [163] and GridWay [85] are designed only to participate in commodity market model. Moreover, they are not designed to trade in the market exchange for leasing resources. Thus, this infrastructure support is needed to provide flexibility to user for trading in any market. Moreover, automatic bidding support is required to participate in auctions used by systems such as Tycoon [104].

Market Exchange

As discussed previously, users and providers can also start negotiation using market exchange's services. The market exchange services are needed to match multiple users to multiple providers. The market exchange systems such as Catnet [54], Bellagio [9], GridEcon [4] and SORMA [121]have restrictive price setting and negotiation policies. In real marketplaces, the choice of negotiation and pricing protocols are decided by participants in the system. This flexibility is critical because the choice of negotiation protocol (auction, commodity market, and one-to-one) and pricing (fixed, variable) can affect the participants utility enormously depending on the current demand and supply.

As the number of consumers and providers grows scalability of the market exchange will become an issue. Thus, some of the components such as job submission and monitoring, which are already well supported by user brokers and meta-schedulers, can be delegated to them. It makes the system more decentralised in the sense that, market exchange mainly act as the middleman for matching users demand to providers supply, and other responsibilities during job submission and execution will be delegated to user and provider's brokers (or RMSs).

A reputation system would also compliment the market exchanges by removing the unreliable and malicious users from the market. In addition to that, marketplaces are needed to be flexible enough to allow the participants to use market protocols of their choice. This feature will require co-existence of multiple negotiations between consumers and providers. This thesis proposes a market exchange architecture to overcome some of

the short-comings of existing systems by allowing co-existence of multiple negotiation of different types.

Core Middleware (i.e., Resource Level Middleware)

Similar to the user level middleware, the existing resource middleware are needed to be extended to participate in market exchange. In addition, these systems support simple job models, and thus more advanced job models such as workflows need to be considered. In addition to that, SLA monitoring is required to ensure that user's QoS satisfaction.

2.7 Summary

In this chapter, a taxonomy and survey of market-based resource allocation approaches and systems are presented. This chapter also provides an overview of the key requirements and components which are needed to be added in the existing Grid middleware in order to support the utility computing. The taxonomy categorises the market-based resource allocation approaches from five different angles: (i) allocation decisions (ii) mechanism's objective (iii) market model (iv) application model (v) participant focus. This taxonomy is then used to classify various Grid middleware, which helps to examine current state-of-the-art in utility Grids, thus identifying gaps which are needed to be filled in.

In the utility Grid, several projects are working to solve issues from both user and provider perspectives. The rapid emergence of utility computing infrastructures such as Amazon's Elastic Cloud, combined with industrial and academic High Performance Computing (HPC) demand, has increased the development of open marketplaces. However, significant work is required to fully benefit from the capabilities of utility Grids. Specifically, the need for coordination between different participants while maximising their utilities has not been addressed. The lack of coordination can lead to underutilisation of resources and starvation of low income users. Thus, market-based meta-scheduling mechanisms are required to tackle such challenges, particularly in situations where a scheduling decision can affect more than just the immediate users and providers. In the following chapters, these issues are addressed in different contexts by proposing "Meta-Broker" architecture which is a market-oriented meta-scheduler. This meta-broker service can be part of a market exchange and interact with user and resource brokers (middleware).

Chapter 3

Market-Oriented Meta-Scheduler Architecture

This chapter presents the architecture of our proposed market-oriented meta-scheduler called "Meta-Broker". First, this chapter motivates the need of such an entity in the utility Grid, and then it discusses the architectural components of the meta-broker. It also presents the high level description of the meta-broker's scheduling protocol, which is considered in this thesis. Finally, to evaluate the advantages of our meta-broker, we compared it with personalised user brokers through a simulation study.

3.1 Motivation

One of the limitations of personalised user brokers for utility Grids, such as Nimrod/G [1] and Gridbus broker [163], is that they can generate schedules which are conflicting with other brokers. Due to the lack of information about other users, each user brokers may end up contending for the same resource. This contention can cause unnecessary delay in the job submissions. To address this problem, we proposed "Meta-Broker" as a middle entity between users and resource providers to enable the coordination and scheduling of applications with conflicting QoS requirements. The meta-broker can simultaneously maximise the utility of each participant while reconciling their QoS requirements. The utility maximisation and reduction in delays, due to the contention for resources, are the main incentive for users to utilise services of the meta-broker.

3.2 Meta-Broker Architecture

The meta-broker acts as a market-oriented meta-scheduler which matches multiple applications of users with multiple resource sites in a Grid. The architecture of the meta-broker

is hybrid in nature i.e. in centralised manner; it allocates the resources to user's applications, while in decentralised manner; it delegates the management of job submission, monitoring and execution to the individual user brokers. The idea behind this is to keep the Grid users and providers, independent of the meta-broker which acts as a matchmaker between them. This architecture provides local autonomy, no centralised control for monitoring and execution of jobs, and distributed ownership to all Grid users and resource owners. More details on the architecture and implementation of the meta-broker within "Mandi" market exchange will be discussed in Chapter 8.

3.2.1 Architectural Components

The high-level architectural components of the meta-broker are shown in Figure 3.1. Users submit their applications for scheduling and execution with QoS requirements using personalised brokers such as Gridbus broker [163]. The meta-broker acts as a matchmaker or clearinghouse between user brokers and local schedulers at various resource sites under different administrative domains.

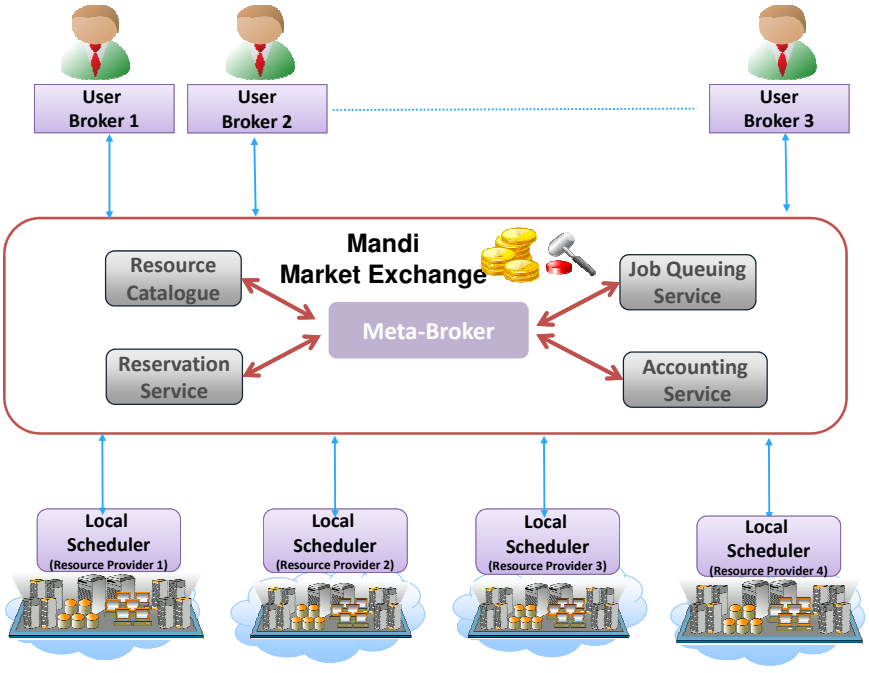


Figure 3.1: Meta-Broker Architectural Components

Resource Provider

Each resource site (cluster, servers, and supercomputer) can be considered as a provider of services such as CPU time slots. Each free CPU slot includes two parameters: number of processors and time for which they are available. All the CPU's at a resource site are

considered to be homogenous while across the resource site CPU configurations may vary. Providers assign CPUs for the exclusive use of the meta-broker through advanced reservation, and supply information about the availability of CPUs and usage cost per second at regular intervals. The economic system considered here is co-operative in nature, that is, the participants trust and benefit each other by co-operating with the meta-broker. Therefore, the possibility of providers supplying wrong or malicious information is discounted. It is assumed that the service price does not change during the scheduling of applications. The scheduling of applications within a resource site is managed by a local scheduler. To reduce fragmentation in resource allocation, the local scheduler at a resource site can use backfilling policies. This can help in maximisation of the resource utilisation. The main functionalities of a local scheduler deployed at a resource site are the following:

- Registration with the meta-broker to participate in the Grid network.
- Update the information regarding the CPU configurations, pricing details, and time slots at a regular interval of time.
- Receive schedule information from the meta-broker.
- Receive resource claims from user brokers and schedule their applications in the agreed time slot.

User Brokers

Using personalised brokers, users submit their application requirements to the meta-broker for resource allocation. Users require their applications to be executed in the most economic and efficient manner. The objective of a user is to have their application, completed by a deadline. It is assumed that deadlines are hard, i.e. a user will benefit only if his/her application is executed by its deadline. Users also provide initial valuations of their applications in terms of budget to the meta-broker. These valuations can be based on the importance of an application to a user or maximum amount the user is willing to pay. The requestor/user broker has following functions:

- User registration with the meta-broker to get resources from the Grid network.
- Resource requests submission and management of the execution as per schedule given by the meta-broker.
- Payment for the resources used for execution.

Application Model Applications considered in this thesis are compute-intensive, and thus, the cost of input and output data transfer is neglected. An application has a fixed number of CPU requirements. We considered the two types of application models:

- **Bag-of-Task model with multiple independent tasks**: Any task of this application can be scheduled on any resource site (Figure 3.2). The execution time of all the tasks is assumed to be same.

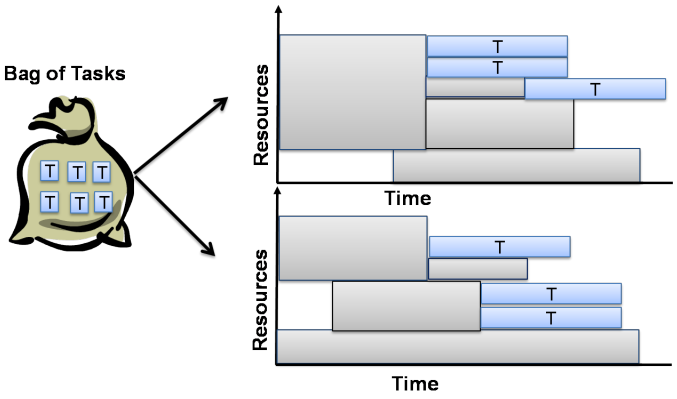


Figure 3.2: Bag-of-Task Application Scheduling

- **The parallel application model with multiple communicating processes**: The application needs to be scheduled within a single resource site in a Grid and all processes need to start at the same time (Figure 3.3). The reason for such a requirement is that the performance of these applications may get severely affected when executed across resource sites with different configurations.

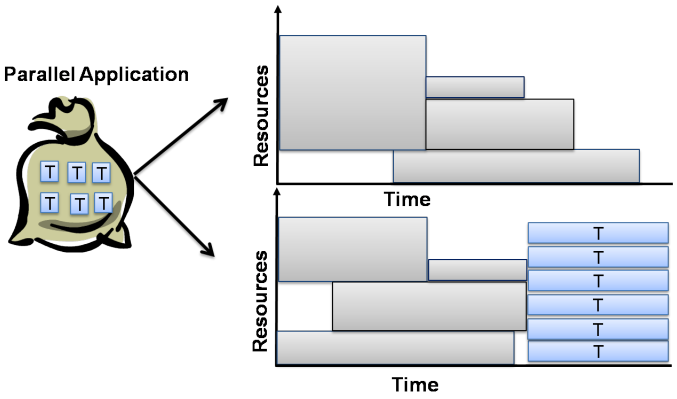


Figure 3.3: Parallel Application Scheduling

To facilitate the comparison of various meta-scheduling algorithms described in this thesis, the estimated execution time of an application is assumed to be known at the time of submission [58]. This estimation can be derived based on user-supplied information,

experimental data, application profiling or benchmarking, and other techniques. Existing performance prediction techniques (based on analytical modelling [124], empirical [40] and historical data [150][89][141]) can also be applied to estimate the execution time of applications. However, in reality, it is not always possible to estimate the execution time of an application accurately. But, in utility Grids where users pay according to the actual resource usage, a user will have to pay more than expected if the execution time of his application has been under-estimated. Thus, a user must still be given the privilege of whether to accept or change any automatically derived estimate before request submission.

Meta-Broker

The meta-broker coordinates the scheduling of user applications on the resources from multiple Grid sites under different administrative domains. It obtains information from any of the registered users and resource providers about their requirements and objectives, and thus takes scheduling decisions.

In addition to provide a common entry point for the registered users to schedule their applications, the meta-broker coordinates competing users while maximising the utility of all the participants. Each resource provider has a local control and ownership of their resources through a local scheduler. The meta-broker does not have control over the set of applications already scheduled to a local scheduler (also referred to as the resource management system). The meta-broker controls and manages the resource allocation decisions whereas job execution and monitoring are managed by the local scheduler.

The meta-broker schedules the applications in a batch mode at the end of a Schedule Interval (SI). At the end of a SI, it calculates the best schedule for all user applications after negotiating available time-slots with the resource providers. The objective of the meta-broker is to schedule all user applications such that their utility is maximised. After allocation of resources to user applications, the submission and execution on the allocated time-slots can be initiated by users [17]. The main functionalities of the meta-broker are the following:

- User and Resource Provider Registration: This creates the certificates and also stores the credentials.
- Collection of all the resource information from the providers.
- Collection of all the resource requests (demand) from users, i.e., broker.
- Scheduling of multiple applications on multiple heterogeneous resource sites.
- Announcement of the scheduling decisions to providers, i.e. local schedulers, and user brokers.

3.3 Resource Allocation by the Meta-Broker

This section illustrates how the meta-broker allocates the resources to users. Figure 3.4 shows the interaction of the meta-broker with the users and resource providers. Each user and resource provider register with the meta-broker. The registration process yields the generation of security keys that are used for negotiation and contract finalisation. The providers periodically advertise to the meta-broker the information about available resources in the form of time-slots. The resource information consists of the number of CPUs available for usage, corresponding load, memory, operating system, Grid middleware used, Million of Instructions per Second (MIPS) of each CPU, and the cost per unit CPU per second. Note that the resource information is dynamic, and the provider can vary any or all of the parameters at any time based on either their local requirements, or based on economy models such as supply and demand. Thus, the meta-broker updates periodically the information about the provider's resources. The users submit their resource requests along with their QoS demands to the meta-broker. The QoS demands consist of budget, deadline, Grid middleware, and preferred/authorised list of resources that the meta-broker should look for, if any. Since the meta-broker itself does not perform the job submission and execution, the authentication and credentials are kept outside the meta-broker's prerogative, and hence, they are sent by resource provider to users after finalisation of schedules by the meta-broker.

The meta-broker runs the scheduling algorithm at periodic intervals so as to satisfy both the users and resources objectives. After matching applications to resources, the meta-broker reserves the resources by contacting each resource provider and issue a Ticket for that. A 'Ticket' contains the execution cost with the reservation information that helps the user broker to schedule an application on the reserved resources.

3.4 Meta-Broker's Internal Control Flow

The scheduling of user applications on multiple resource sites is enabled through interaction of the meta-broker with other components of market exchange as shown in Figure 3.1. This section presents the flow of interaction between each component during the meta-scheduling of user applications. The *Meta-Broker* is the key component for enabling coordinated and efficient resource allocation. Its interaction with other components involves following steps:

1. The information about available resources are stored in the *Resource Catalogue*.

2. The users' resource requests with QoS requirements are queued using *Queuing Service*.

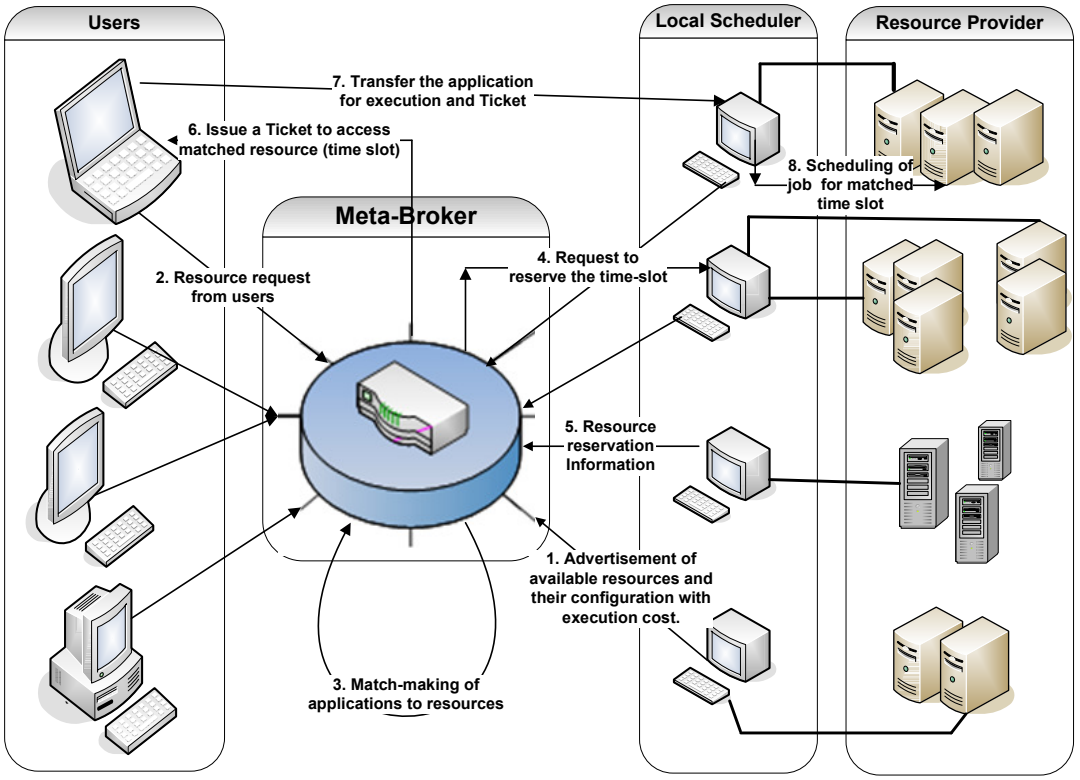


Figure 3.4: Meta-Broker Protocol

3. The *Meta-Broker* periodically gets all the unmatched resource requests from the *Queuing Service*.

4. The *Meta-Broker* makes decisions of allocating resource based on the users' QoS requirements and available resources in the *Resource Catalogue*.

5. The *Meta-Broker* passes all the scheduling/matching decisions to the *Reservation Service* where a Ticket is issued for each reservation.

6. The *Meta-Broker* passes these Tickets to the respective users.

7. The *Meta-Broker* then updates the account of all the participants using *Accounting Service*.

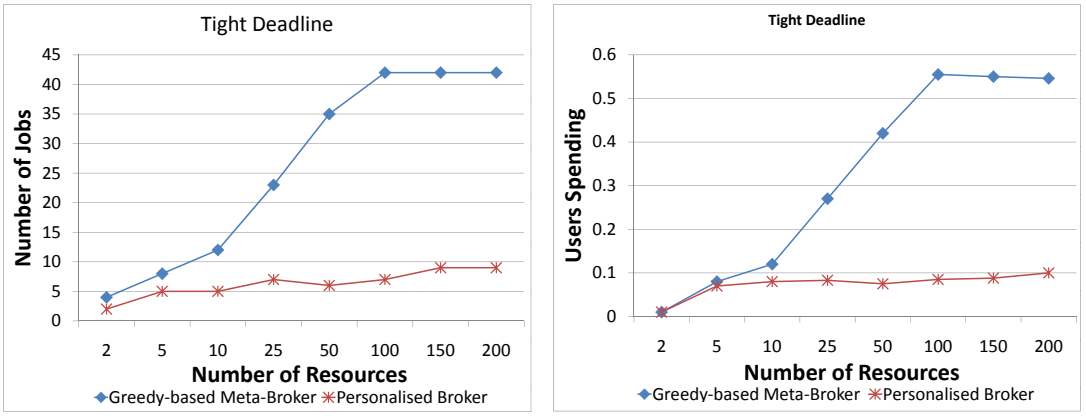
3.5 Comparison between Personalised and Meta-Broker

In this section, we compare our proposed meta-broker with the completely decentralised system, where each user has his/her personalised broker for scheduling on utility Grids. To study their performance, we used the greedy-based scheduling policy on both the systems, to minimise the user spending for application execution. In the greedy-based scheduling

policy, each of the brokers tries to schedule an application on a resource provider with the lowest price.

We simulated the utility Grid with heterogeneous resources having different MIPS [80] ratings and each resource site having different number of CPUs in the range of 4 to 12 with average number of CPUs as 8. A utility Grid having resource sites in the range of 25 to 200 is generated. The usage cost per second of each resource is varied (using Gaussian distribution) between 4G$-5G$ (G$ means Grid dollar) with average cost of 4.5G$. Execution time in the simulator setup for the several hundred runs restricts us to model 50 jobs per run. Thus, the meta-broker schedules the jobs (which consist of several tasks, each require one CPU to run) submitted by 50 concurrent users with schedule interval of 50 seconds. The jobs are also simulated with varying the QoS requirements. Jobs with an average number of 5 tasks, and having 10-50% variations in the number of tasks (using Gaussian distribution) are considered, and all the jobs are submitted within 20 seconds of the simulation start time. Average estimated run time for the jobs is also taken to be 400 seconds and varied by 20% using Gaussian distribution. All the jobs are given relaxed budget constraints (i.e., twice the cost of average job execution time). The simulations are performed for different deadline scenarios as listed below:

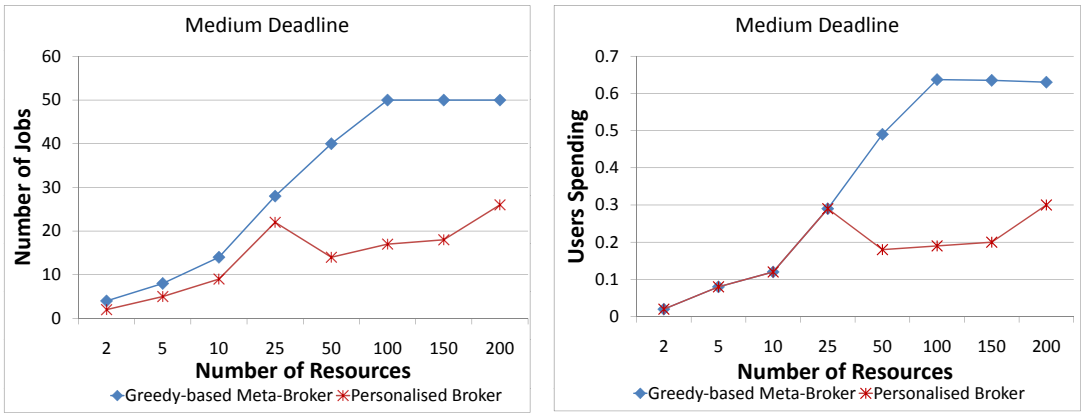
- Experiment-1: Tight deadline (estimated time+(50 seconds with 20% variation)).
- Experiment-2: Medium deadline (estimated time+(250 seconds with 20% variation)).
- Experiment-3: Relaxed deadline (estimated time+(500 seconds with 20% variation)).



(a) (b)

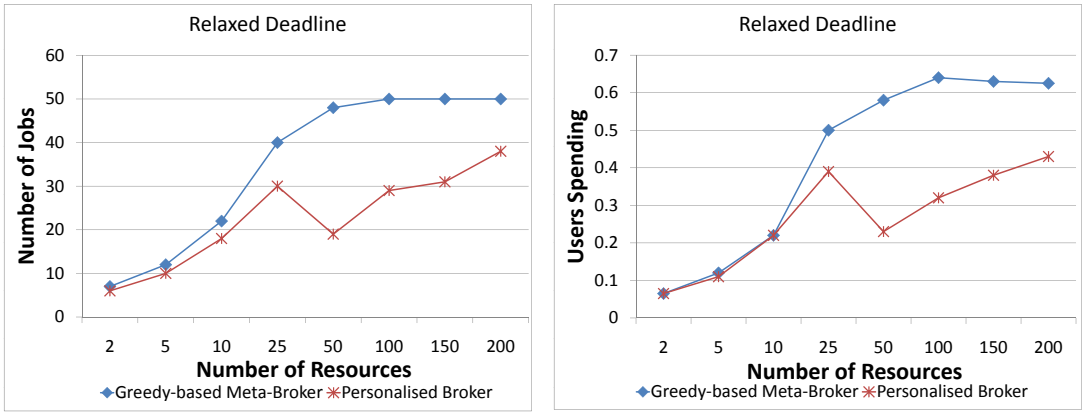
Figure 3.5: Effect of Tight Deadline on Users

3.6 Performance Results



(a) (b)

Figure 3.6: Effect of Medium Deadline on Users



(a) (b)

Figure 3.7: Effect of Relax Deadline on Users

Figure 3.5(a), 3.6(a), and 3.7(a), shows how jobs completed (out of the 50 concurrent jobs submitted) by the two types of brokers vary with the number of Grid resources. In the second set of results, i.e., Figure 3.5(b), 3.6(b) and 3.7(b), aggregated revenue earned for the completed jobs is shown. Figure 3.5(b) clearly indicates that in spite of having more resources, the personalised broker could not execute all the jobs. For example, when 200 Grid resources were available the personalised broker could only complete 9 jobs out of the possible 42 jobs (8 jobs had very tight deadline, hence cannot be completed within the deadline provided), whereas in the case of Greedy-based meta-broker all the jobs were finished when the nodes available were around 50 or more. Similar trend can also be noticed in the case of Medium (Figure 3.6) and Relaxed deadline (Figure 3.7).

The reason behind this is the lack of coordination between personalised brokers. Thus, when a user tries to establish a contract or SLA, the resource owner would select only the first user and would send a fail SLA for all other users. Failed-SLA jobs are re-submitted by the personalised broker that would result in the same situation with reduced users as compared to previous schedule, provided no other user enters the Grid. The process would be repeated until the job eventually finds a resource where SLA is established. Meanwhile, if the deadline is over, then the job is automatically thrown out. This is a live lock situation, where, even if the resources are available the jobs get omitted.

3.7 Summary

This chapter describes the meta-broker architecture and resource allocation model, which enables the meta-scheduling of multiple user applications on the resources under different administrative domains. We also evaluates the advantages of our proposed meta-broker architecture over personalised user brokers through a simulation study. To achieve the goal of utility maximisation for Grid participants, various market-oriented meta-scheduling mechanisms in the three scenarios are designed and presented in Chapter 4 to 7.

Chapter 4

Meta-Scheduling to Minimise User Spending

This chapter describes the market-oriented meta-scheduling algorithms to reduce users' spending. The user applications are constrained by QoS parameters such as deadline and budget. To investigate the challenges in this meta-scheduling problem, we present a Linear Programming/Integer Programming (LP/IP)-based analytical model of meta-scheduling of applications with deadline and budget constraints. The LP model is then used to design a Genetic Algorithm (GA), which minimises the combined cost of all users participating in the meta-scheduling. Finally, a simulation-based study is presented to show the effectiveness of the proposed algorithms in minimising the combined spending of all users.

4.1 Problem Definition

As discussed in the previous chapter, users and resource providers interact through the meta-broker. Grid users submit their applications to the meta-broker with their QoS requirements. The QoS requirements consist of budget, deadline, and number of CPUs required. Each application consists of independent tasks with each task requiring one CPU. Grids, such as Grid5000 [28], are generally composed by many resource sites, where each site consists of homogeneous CPUs. The following two kinds of user applications are modelled:

1. Multi-Grid node (MGN) applications can split into independent tasks, which can run on different resources. For example, bag-of-tasks applications.

2. Single-Grid node (SGN) applications that require a complete set of CPUs on the same resource and fail if all CPUs are not available. For example, synchronous parallel applications.

The meta-broker gathers the information from resource providers such as CPUs available, Grid middleware, cost of CPUs per unit time for each user and CPUs capacity (e.g. Millions of Instruction per Seconds (MIPS) rating). The meta-broker generates a combined schedule based on algorithms discussed in subsequent sections. As the demand for resources may exceed the supply in the Grid, some applications are placed in the queue for later assignment when resources are available or freed.

4.1.1 Problem Formulation

The objectives of meta-scheduling in the above environment with m resources and n applications are as follows:

- Scheduling based on QoS, i.e., budget and deadline.
- Minimise combined cost for all the users by meeting the budget and deadline constraints.
- Maximise the total number of users' application submission.

Let $P_i(n_i, c_i, v_i)$ represents the information that the meta-broker receives from Grid resource site i at a given instance T, where n_i is the number of free CPUs available, c_i is the cost of using a single CPU per second for application/user j, v_i is the MIPS speed of one of the CPUs, and $i \in I = 1...m$. Let Q_j represents the QoS requirement that the meta-broker receives from user j. Let b_j, d_j, M_{kj} and m_j be the budget constraint the user specifies, the deadline constraint for finishing the application, the size of each task in the application in terms of Millions of Instructions (MI) [80], and the number of CPUs the user requires for executing the application respectively. Let $j \in J = 1..n$, and $k_j \in K = 1..m_j$.

The mathematical model for scheduling of SGN applications is as follows:

$$c = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_i x_{ij} r_{ij} Max_k(M_{kj}/v_i) \tag{4.1}$$

In Equation 4.1, the variable r_{ij} denotes the number of CPUs allotted to application j on resource i. Note that, in the case of SGN applications, the value of r_{ij} would be either 0 or m_j (the number of CPUs required by the application j). The variable x_{ij} denotes that whether application j is mapped to resource i or not. (M_{kj}/v_i) refers to the execution time of task k_j on resource i. Equation 4.1 denotes the cost minimisation function that

represents the dollar cost spent by all the users. The constraints are following:

$$\sum_{j=1}^{n} r_{ij}x_{ij} \leq n_i, \forall i \in I \tag{4.2}$$

$$\sum_{i=1}^{m} r_{ij}x_{ij} = m_j, \forall j \in J \tag{4.3}$$

$$x_{ij} \in \{0,1\}, \forall i \in I, j \in J \tag{4.4}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \forall j \in J \tag{4.5}$$

$$\sum r_{ij}c_{ij}Max_k(M_{kj}/v_i) \leq b_j, \forall j \in J \tag{4.6}$$

$$Max(M_{kj}/v_i)x_{ij} \leq d_j, \forall i \in I, j \in J, k \in K \tag{4.7}$$

$$c_i, x_{ij}, r_{ij}, b_j, \ and \ d_j \geq 0, i \in I, j \in J \tag{4.8}$$

Equation 4.2 denotes the resources capacity constraints. Equation 4.3 denotes the CPU requirement of an application i.e., m_j. Equation 4.4 and 4.5 denote the assignment matrix, which indicates that an application can be assigned to a maximum of one resource. Equation 4.6 is added for satisfying the budget constraints and Equation 4.7 is added for the time constraints. Equation 4.1 is solved for minimisation to obtain an optimal cost efficient schedule. All the quantities, i.e., the resource parameters, cost, deadline and budget should be greater than zero, hence the Equation 4.8. Note that Equation 4.4 and 4.5 make the problem NP-hard as these constraints make the problem a 0-1 assignment problem, which is well known to be NP-hard [112]. LP/IP model for scheduling MGN applications is almost same as for SGN applications. The only difference is that model for MGN applications will not include constraints 4.4 and 4.5.

4.2 Proposed Algorithms

4.2.1 Linear Programming-based Algorithm for Scheduling MGN Jobs

As discussed in the previous section, the mathematical model for MGN application scheduling will be converted to an LP/IP model after removing constraints 4.4 and 4.5. LP/IP can be then easily solved optimally using any standard integer programming algorithms such as Simplex algorithm. There are many open-source and commercial implementations of such algorithms. For simulation purposes, we have used the Coin-OR library [138]. The Coin-OR library is a high quality open-source LP solver, and its main strengths are its Dual and Primal Simplex algorithms.

4.2.2 Linear Programming-based Algorithm for Scheduling SGN Jobs

Due to the absence of constraints 4.4 and 4.5, LP/IP solutions obtained in Section 4.2.1 are infeasible for the SGN application scheduling problem. Thus, we need to address the problem in the two phases: a) approximation of LP/IP solution to a feasible solution b) search for the optimal solution. For the first phase, we have designed a novel algorithm Modified MinCost (MMC) to approximate an infeasible LP/IP solution to a feasible solution for the SGN application scheduling problem. For the second phase, we have used a GA-based heuristic to further optimise the feasible solution. We have chosen GAs for solving budget and deadline constraint cost minimization problem mainly due to two reasons. First, GAs are more robust than standard heuristics for assignment problem. If good initial seed and enough computation time is given, then GA most likely approximates a global optimum, while heuristics in many cases are trapped in a local optimum. Second, GAs performance is evaluated in many previous studies such as by Golconda et al. [72] who evaluated performance of five heuristics. It was found in their study that GA produces the best utility for users in comparison to the heuristics such as Min-Min. Thus, we have used GA in our second phase. The details about the chosen strategies in each of these phases are explained in the following sections.

Modified MinCost (MMC) Algorithm

Our MMC algorithm is inspired from 'Minimum Cost Algorithm' which is a well-known method for finding the feasible solution near to the optimal in transport problems. In the Minimum Cost algorithm, we find the minimum cost resource, assign the maximum number of applications to that, and reduce the demand and supply. The process is repeated until all applications are allocated.

In MMC, we take an optimal solution from the LP-based algorithm (discussed in the previous section), and then, approximate it to obtain a feasible solution for SGN scheduling problem. The solution obtained from this algorithm acts as an initial solution to the GA.

The pseudo code for MMC is given in Algorithm 4.1, which shows how MMC approximates a LP/IP solution. It takes as an input a list of applications which is a data-structure containing information regarding, how different tasks of an application are mapped to resource providers, and how many CPUs are required by the application. This application list is generated from the algorithm for scheduling MGN applications. First, MMC algorithm makes a copy of the input, and then sorts application list on the basis of the number of resources to which applications are mapped (line 1-2). This is done so that we get a solution as closer as possible to the optimal LP/IP solution.

We handle differently the applications that are mapped only to (a) one resource and (b) more than one resource. For applications which are mapped only to one resource,

we freeze their mappings. The mappings will not change during the algorithm, and thus we reduce the total capacity of the resource (line 4, 5, 6). Then, these applications are added later to the schedule list (line 7). The applications, which were mapped to more than one resource, will be re-mapped to a resource p_i which has the capacity to allocate the full application $j1$ and is the most economical (line 9-11). Other applications which are allocated to this resource will be re-mapped to resource where application $j1$ was allocated other than p_i, if deadline and budget constraints are satisfied; otherwise they are allocated to dummy resources (line 15). Other applications, which are not allocated in the above process, will be mapped to the dummy resource (line 19). These applications are re-mapped to real resources using a greedy approach.

Algorithm 4.1: ModifiedMinCost (ListJobs)

```
1  ListJ← Listapplications.clone()
2  sort(ListJ)
3  foreach Job j1 in ListJ do
4      if numofProviders(j1)=1 then
5          changeAvailableCapacity(Provider(j1),j1)
6          Goto step 20
7      end
8      else
9          P← J1.providerlist.clone()
10         sort(P)
           // number of CPU allocated on Provider
11         foreach provider in P do
12             if remainingCapacity(p)>PERequired(j1) then
13                 AllocateFull(j1,p)
14                 InterchangeCapacity(p,p.applicationlist[])
15                 Goto step 20
16             end
17         end
18     end
19     addapplication(j1,DummyResProv)
20 end
21 ScheduleDummyResJob()
22 MakeScheduleList(ListJ)
23 return ListJ
```

Complexity: Complexity of MMC algorithm for m resources and n applications is $O(nlogn + mn + O(interchangeCapacity()function) * n)$. We have taken a simple implementation of *interchangeCapacity* function using arrays, which has worst-case complexity as $O(mn)$. Thus, the worst-case complexity of the algorithm becomes $O(mn^2)$, which is very high upper bound over the real complexity as average number of applications per resource will be less than the total number of applications.

Genetic Algorithm formulation

For the second phase, we have used GAs [34]. GAs provide a robust search technique that allows a high-quality solution to be derived from a large search space in polynomial time, by applying the principle of evolution. A GA combines the exploitation of best solutions from past searches with the exploration of new regions of the solution space. Any solution in the search space of the problem is represented by an individual (chromosomes). A GA maintains a population of individuals that evolves over generations. The quality of an individual in the population is determined by a fitness-function. The fitness value indicates how good the individual is compared to others in the population. A typical GA consists of the following steps:

1. Create an initial population consisting of randomly generated solutions.
2. Generate new offspring by applying genetic operators, namely selection, crossover and mutation, one after the other.
3. Compute the fitness value for each individual in the population.
4. Repeat steps 2 and 3 until the algorithm converges.

Convergence can be a different criterion for different problems, but generally the absence of changes in the solution for n generations is considered as convergence. The n could be application specific again; in our implementation n is assigned a value of 10. The following space encoding and fitness function used in our GA-based heuristic:

- **Chromosome Encoding**

  The solution for the scheduling is an assignment vector S ($= s_1, s_2, ..s_j, .., s_n$), where each element s_j in the vector represents the node/resource site onto which the application is scheduled. The vector S represents a chromosome (individual) in the population. Since each application can be assigned to a maximum of one node, it can be denoted as a simple single dimensional integer vector of size equal to the number of applications. Note that the node number s_j can be the same for multiple applications, which means that, if there are sufficient resources then the node can execute multiple applications.

- **Fitness Function**

  The fitness function for our GA is a cost function on the same lines of Equation 4.1 and is denoted as follows:

$$F(S) = \sum_{j=1}^{n} \sum_{k=1}^{m_j} c_{s_{ji}j} * M_k / v_{s_j} + \alpha_j, \; where \; \alpha_j = C * p_j \tag{4.9}$$

α_j represents the QoS index; p_j is the priority of application *j* (priority sequence number, i.e., *(1..n)*); and *C* is a cost constant chosen such that the fitness value of a least priority (higher value of p_j) application assigned to real node is always less than a higher priority (lower value of p_j) application scheduled to a dummy node. The use of p_j into the fitness function gives preference to the applications which promise better QoS; and constraints on C ensure that the solution does not prematurely end in a local minimum.

The computation of QoS index α_j is discussed in the next section. If constraints listed in Equation 4.2, 4.5 and 4.6 are not satisfied then an infeasible cost or very high cost is assigned to S.

- **QoS index**

  The QoS index is used to assign priority to applications. Generally priority is a personal or providers preference, but in an ideal market situation the priority is linked with QoS, i.e., priority is given to the users who are willing to pay more incentives, and to provide relaxed deadlines. Therefore it can be represented as follows:

$$\alpha_j = C * p_j,\ where\ p_j = b_j / Max(1, d_j - t_j) \tag{4.10}$$

  In the above equation, C is a constant value, and t_j is the execution time of application j. The QoS index is a fairness index computed for each application based on their deadline and budget parameters. Ideally the user pays either the entire budget or amount proportional to its priority. As we are considering a fair market meta-broker scheduling, where user pays only for the resources he/she consumed, the priority index is computed by using following equation.

$$if(d_j > (T + t_j)),\ \alpha_j = 1/MAX(a, d_j - (T + t_j))\ else\ \alpha_j = 0 \tag{4.11}$$

  T is the current schedule time. This function assigns priority to the applications/users according to their finish time and incentive offered. Note that as the time increases, even the applications with relaxed deadlines gain priority and similarly, if the user provides a tight deadline then there is a danger of getting eliminated from scheduling very quickly.

Linear Programming-based Genetic Algorithm (LPGA)

LPGA seeds the approximate LP solution from MMC (first phase) with a GA (second phase) for generating a solution near to the optimal solution. Thus, first we solve a LP/IP problem without constraints 4.4 and 4.5 and then MMC approximates the solution ob-

tained in the previous step. Finally, using the feasible solution from MMC, we iterate various genetic operations to get the best solution.

The pseudo code is given by Algorithm 4.2. As the scheduling is done by the meta-broker periodically, it waits for the applications from the users (taken as QoS requirement for the application in line 4) and the resource load from the provider (line 3). Then, we have to add a dummy application queue with high cost and number of CPUs to make scheduling problem balanced, i.e. to balance the supply of resources and the demand by users for resources (line 6-8). After that, we sort the resource list on the basis of their cost, and the application list on the basis of QoS index (line 11-13). After sorting the resource list and the application list, the LP/IP solution is generated that is obtained without considering constraints 4.4 and 4.5 discussed in the Section 4.1.1 (line 15). This solution is approximated using the MMC algorithm (line 16) which is used to generate the initial population space for the GA (line 20-23). From the initial population space, we select the best chromosome values on the basis of the fitness function. These best chromosomes are then mutated to generate the next generation population (line 24-35). This operation is repeated until the convergence criterion is reached or the maximum specified limit of iteration is reached (line 26). Based on the chromosome, the schedule list is generated and users are notified about mappings (line 39-41).

One limitation to the problem formulation is that if the number of resources requested is more than the number of resources available for the broker, then the problem results in an infeasible solution. Therefore, we propose the introduction of dummy resources with infinite capacity and having more cost than any of the resources available on the Grid. This enables the algorithm to converge and assign some of the applications to the dummy resources. The fitness function forces the applications with lower priority to be scheduled on the dummy nodes rather than the applications with higher priority. Dummy resource applications are rolled over to the next schedule period, as they cannot be executed.

4.3 Performance Evaluation

4.3.1 Simulation Methodology

Workload Characteristics: The meta-broker schedules the jobs (which consist of several tasks, each require one CPU to run) submitted by 50 concurrent users with simulation interval of 50 seconds. Execution time in the simulator setup for the several hundred runs restricts us to model 50 jobs with varying QoS requirements per run. The jobs characteristics such as size and runtime are generated using random distributions. We modelled jobs with average number of 5 tasks, and having 10-50% variations in the number of tasks (using Gaussian distribution), and all the jobs are submitted within 20 seconds of simula-

Algorithm 4.2: Algorithm 2. LPGA()

```
1  Schd_List=null
2  while current_time <next_schedule_time do
3      RecvResourcePublish(Pj)
       // from providers
4      RecvJobQos(Qj)
       // users
5  end
6  if numJobs>numResource then
7      AddDummyNode()
8  end
9  ListP=ListProviders.clone()
10 Sort(ListP)
   // cost of resource
11 foreach application 'j' in Q do
12     compute QoS index
13 end
14 ListJ=sort(Q)
   // descending order of QoS index
15 LPschedule=GetLPsolution()
   // from Coin Library
16 ScheduleList= ModifiedMinCost(LPschedule)
17 AddinPopulation(POPU_LIST, ScheduleList)
18 generateNextGenerationPopulation()
19   begin
20     foreach i in (population_size-POPU_LIST.size()) do
21         cromos=GenerateRandomCromosome(i)
22         POPU_LIST.add(cromos)
23     end
24     ComputeFitnessfunction()
25     bestCromos=SearchBestCromosome(POPU_LIST)
26     if termination then
27         return bestCromos
28     end
29     doSelection()
30       begin
31         SelectBestCrossoverRateCromosome()
           // 'based on'Roulette wheel selection policy
32         DoCrossover()
33         DoMutation()
34         AddtoPopulationList(POPU_LIST)
35         Goto step 18
36     end
37 end
38 Schd_List =GenerateSchedulelist(bestCromos)
39 foreach element in Schd_List do
40     notifyuser();
41 end
```

tion start time. The average estimated run time for the jobs is also taken to be 400 seconds and varied by 20% using Gaussian distribution. Even though runtime of jobs can be varied other than 20%, we believe that similar results can also be expected in other cases. In addition, the variation in runtime is chosen to analyze the effect of jobs with similar needs. All the jobs are given relaxed budget constraints (i.e., twice the cost of average job execution time), so that it is always greater than job execution cost. The job execution cost can be computed by using resource_usage_price*execution_time*number_of_CPUs.

Configuration of Resources: We considered the Grid with heterogeneous resources having different MIPS [80] ratings, and each resource having different number of CPUs in the range of 4 to 12 with mean number of CPUs as 8. Grid resources are varied from the range of 25 to 200. Since, there is no information available on how to model usage cost of a resource, thus, the usage cost per second per CPU unit of each resource is synthetically generated. The usage cost is varied (using Gaussian distribution) between 4G$-5G$ (G$ means Grid dollar) with mean cost of 4.5G$.

Experimental Scenarios: To comprehensively evaluate the performance of our algorithms, we examine three experimental scenarios for different job deadlines i.e. tight, medium and relaxed. The deadlines are model based on how much users are willing to wait to have their job executed. The tight deadline is modelled as very urgent scheduling requirement from users. Thus, the tight deadline is set to $job's\ execution\ time + scheduling\ interval\ time$ (i.e. 50 sec). The user having medium deadline is modelled considering that he is willing to wait for at least 50% of his/her job's execution time. Thus, the medium deadline is set to $150\%\ of job's\ execution\ time + scheduling\ interval\ time$ (i.e. 50 sec). Similarly, the user having relaxed deadline is willing to wait more than the job's execution time. Thus, the relaxed deadline is set to more than $200\%\ of job's\ execution\ time + scheduling\ interval\ time$ (i.e. 50 sec). Since, in our experiments, the average execution time of an job is set to 400 sec, therefore the following experiments with different deadlines are performed

- Experiment-1: Tight deadline (estimated time+(50 seconds with 20% variation)).
- Experiment-2: Medium deadline (estimated time+(250 seconds with 20% variation)).
- Experiment-3: Relaxed deadline (estimated time+(500 seconds with 20% variation)).

Our performance evaluation examines the relative performance (users spending and number of jobs completed) of LPGA with respect to three other following meta-scheduling algorithms for the above type of jobs by varying the number of resources:

1. Greedy-based meta-scheduling algorithm.

2. Heuristic GA algorithm (HGA) with initial seed from Greedy-based algorithm.

3. Modified MinCost algorithm (MMC).

The results for the three different simulation scenarios for both MGN LP-based algorithm and SGN LP driven Genetic Algorithm (LPGA) are discussed in the next section.

4.3.2 Performance Results

LP/IP-based Meta-Scheduling Algorithm for MGN Jobs

Table 4.1: Total Cost Spent by Users for MGN Jobs

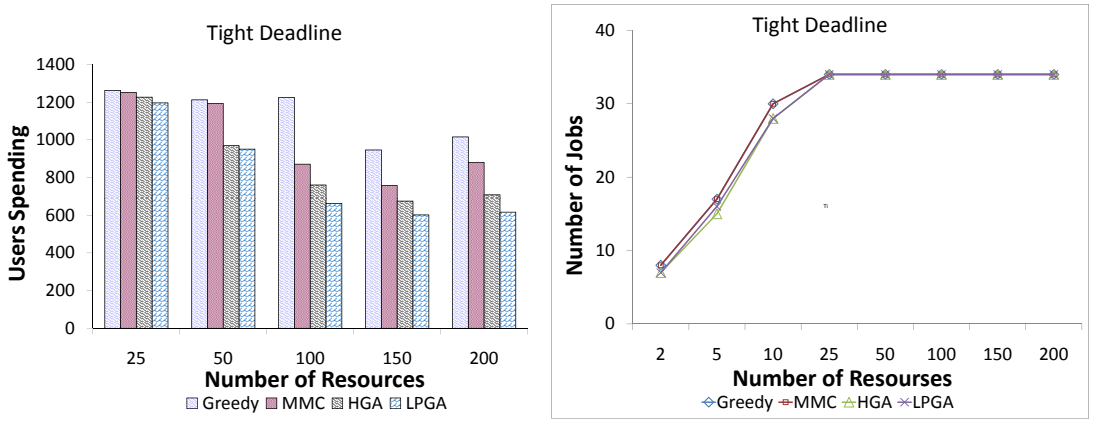
Total Number of Providers	Cost in Medium Deadline	Total Cost in Relaxed Deadline	Total Cost in Tight Deadline
25	259832.6	194668.9	245280.8
50	261059.7	198771.5	247655
100	260689.1	198462.5	247251.2
150	260653.5	198360.5	247245.4
200	260664	198455.3	247015.7

The comparison results of our proposed algorithm LPGA with other meta-scheduling algorithms for MGN jobs are compiled in Table 4.1. In tight deadline out of 50 applications (246 tasks) only 34 applications (170 tasks) were scheduled where as in medium and relaxed deadlines all the applications i.e. 50 (246 tasks) were scheduled. The reason behind this is the tight deadline, for which many applications missed the deadline constraints due to simulation interval. Thus, even though the number of resources is increased, the number of scheduled applications remains the same. Moreover, in the case of relaxed deadline, the users spending is the minimum in comparison to the other deadline types because more applications are scheduled to resources with the least cost. Also, even though it seems that this trend is not followed when we compare tight deadline and medium deadline scenarios, more number of applications with medium applications are completed. It is also interesting to note that in the tight deadline scenario, with the increase in the number of resources; the total cost spent by users is also decreasing.

SGN Job Scheduling Algorithms

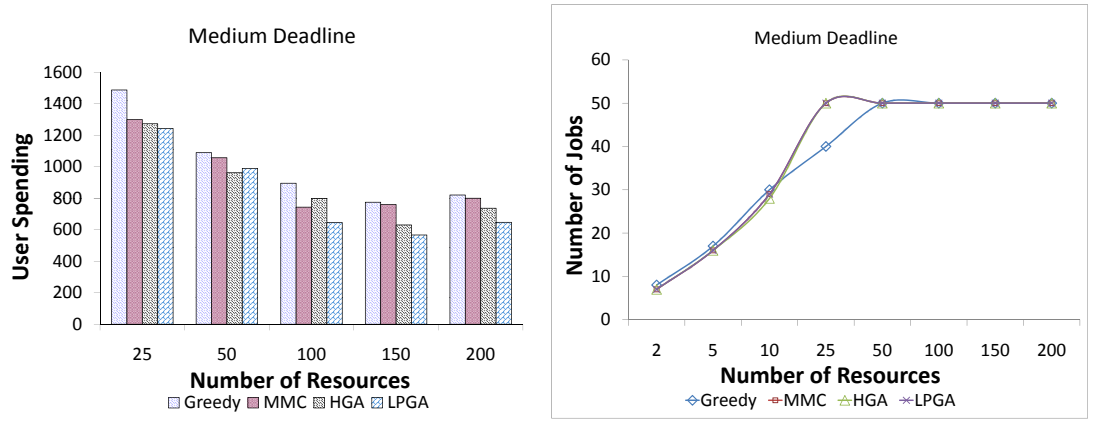
In this section, the performance of four meta-scheduling algorithms has been compared using the number of applications scheduled and the users spending. The results are compiled and presented in Figure 4.1 to 4.3. In the first set of results, i.e., Figure 4.1(a), 4.2(a) and 4.3(a), aggregated revenue earned for the completed applications are plotted on

Y-axis. In the first set of results, i.e. Figure 4.1(b), 4.2(b) and 4.3(b), X-axis presents the Grid resources while Y-axis presents the applications completed (out of the 50 concurrent applications submitted). Figure 4.1(a), 4.2(a) and 4.3(a) for relaxed, medium and tight deadlines show that LPGA has outperformed all other meta-scheduling algorithms, thus minimising the users spending. In case of all the deadline, even though initially, the user spending in schedule given by HGA and LPGA seems to be same, as number of resources increases, LPGA decreases the users spending if compared to other algorithms. Since the cost variations within the Grid resources are not significant (i.e. 4.5 G$ with 0.5G$) only up to 7-8 % cost benefit was noticed. However, more benefit can be anticipated if the variations are higher.



(a) (b)

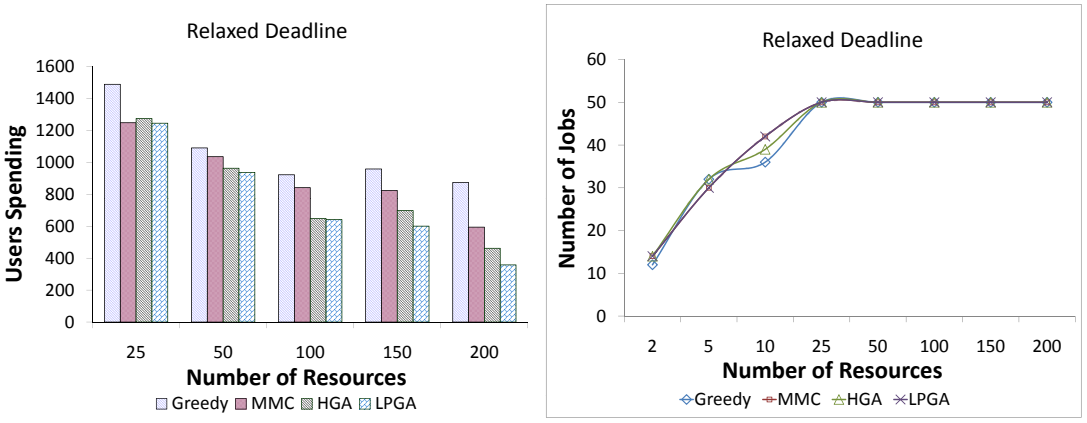
Figure 4.1: Effect on User Applications with Tight Deadline



(a) (b)

Figure 4.2: Effect on User Applications with Medium Deadline

More interesting results can be observed when we compare the performance of MMC algorithm and the greedy approach. It is clear from the graphs that the cost gain for users



(a) (b)

Figure 4.3: Effect on User applications with Relaxed Deadline

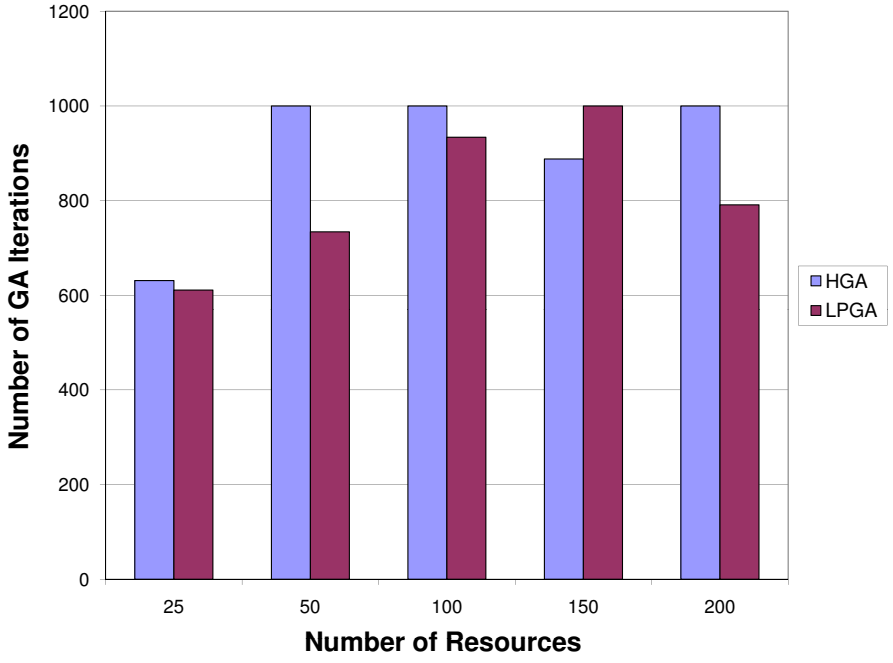


Figure 4.4: Comparison of Number of Iterations in HGA and LPGA

in the case of MMC algorithm can be as much as 30% than greedy approach. Thus, the results demonstrate the users spending for each application decreases when LPGA is used by the meta-broker for combined scheduling of applications. This is because MMC is giving the better schedule which is more closer to the optimal than the greedy algorithm.

It can be noticed from Figure 4.1(b), 4.2(b) and 4.3(b) that the curves become smoother and peak early compared to tighter deadline. This trend is due to the relaxed deadline having more applications completed with the same number of resources. As when the deadline is more relaxed, the meta-broker tries to complete more and more tasks. With increase

in the number of resources, all algorithms schedule the same number of user applications. In case of tight deadline (Figure 4.1(b)), many applications missed the deadlines (very tight); only 34 applications are completed even though the number of resource increased to 200.

The next simulation study presents the advantage of LPGA over HGA algorithm. We have compared the number of iteration taken by both algorithms in case of tight deadline. The results shown in Figure 4.4 indicate that in case of LPGA, the number of iterations is reduced with a decrease in user spending. For example, with 50 resources, the number of iterations performed in HGA is 1000, whereas in LPGA it is less than 700. Even though the number of iterations in HGA is less than in LPGA with 150 resources, the revenue generated is smaller which can be observed from Figure 4.1(a). These results also indicate that in spite of taking more iteration, the HGA could not find the global/better optimum. The reason for this is the property of GA-based heuristics which are search algorithms. The convergence and search domain of a GA are highly dependent on the seeded population. If the seed is in a domain where the sub-optimal solution exists but the seed is far away from sub-optimal solution, then the GA may require more number of iteration to reach the sub-optimal solution. Hence, to search a better solution (either the one found by LPGA implementation or any other better solution) the algorithm would have required a higher number of iterations. This clearly explains the reason for a higher number of iterations in the case of LPGA when the number of resources is 150.

4.4 Related Work

Many works have investigated the application of market models and mechanisms to improve scheduling in Grids. The most relevant works in the context of maximising the user's utility are [25][169][61]and [46]. The Gridbus broker [25] is a personalised broker that uses a greedy approach to schedule a parameter sweep application with deadline and cost constraints. We differ from them in our aim that is to schedule multiple applications having different QoS requirements to optimise combined cost.

G-commerce [169] is another economic-based study that applies strategies for pricing Grid resources to facilitate resource trading, and compares auction and commodity market models using these pricing models. Feng [61] proposed a deadline cost optimisation model for scheduling one application with dependent tasks. There are some limitations of these studies: (1) the algorithms proposed are not designed to accommodate concurrent users competing for resources (2) the application models considered are simple i.e. independent task or parametric sweep application. Similarly, Dogan et al. [46] proposed a meta-scheduling algorithm considering many concurrent users with single task applications. On the other hand, in our work, we have modelled both independent and parallel

applications submitted by concurrent users in a meta-scheduling environment to minimise the combined cost of all users.

In the Grid environment, where each user has different QoS constraints, the scheduling problem becomes a Generalised Assignment Problem (GAP) that is a well known NP hard problem. GAP can be solved using GAs [34]. Thus many GA-based heuristics are proposed in the literature. Wiessman et al. [98] proposed a novel GA-based algorithm that schedules a divisible data intensive application. Martino et al. [45] presented a GA-based scheduling algorithm where the goal of super-scheduling was to minimise the release time of applications. These GA-based heuristics do not consider the QoS constraints of concurrent users such as budget and deadline. The viability of an evolutionary algorithm-based scheduling such as GAs for realistic scientific workloads is demonstrated by systems such as Mars [13]. Mars is a meta-scheduling framework for scheduling tasks across multiple resources in a Grid. In a recent study [72] where five heuristics were compared, it was found that the GA produces the best utility for users in comparison to the heuristics such as Min-Min. Thus, for comparison, we have used GA seeded with greedy as an algorithm. In this work, we have used LP/IP model to solve the cost minimisation scheduling problem in Grid. The LP/IP can offer optimal solutions in minimisation/ maximisation scheduling problems when constraints and objective functions are linear. There are many works on scheduling in domains other than Grid using LP/IP model. In Feltl et al. [60], an LP/IP-based initial solution and followed by an intelligent replacement of offspring based on Martello et al. [112] is proposed. But these models do not consider the deadline and budgets constraints. As they are developed based on specific domain knowledge, they cannot be applied directly to Grid scheduling problems, and hence. have been enhanced accordingly.

4.5 Summary

This chapter models the scheduling of concurrent user applications in utility Grids as an LP/IP problem. The meta-broker maximises the utility of all the users participating in the meta-scheduling. We have presented a novel algorithm called MMC that approximates the LP solution to derive a near-optimal solution for single-grid node application scheduling. LPGA is designed to decrease the combined user spending and increase the resource utilisation by seeding the solution from MMC algorithm to the GA. We have also taken into consideration user QoS constraints, i.e., deadline, number of CPUs and budget. The results show that LPGA outperforms other meta-scheduling algorithms such as the greedy approach and HGA by giving the minimum monetary cost.

In this chapter, we have considered only one objective function i.e. cost minimisation, but in some scenarios, users may want to optimise both response time and execution cost

of the application. Thus, the user has to choose between multiple conflicting objectives, which makes the problem of meta-scheduling more challenging. In the next chapter, we will show how this challenge can be tackled using a “trade-off metric”.

Chapter 5

Meta-Scheduling to Minimise Time and Cost for Users

This chapter proposes three meta-scheduling heuristics, i.e. MinMin Cost Time Trade-off (MinCTT), Max-Min Cost Time Trade-off (MaxCTT) and Sufferage Cost Time Trade-off (SuffCTT), to manage the trade-off between the overall processing time and monetary cost. Thus, these heuristics aim to minimise simultaneously both the response time and monetary cost on the basis of a trade-off factor. The trade-off factor indicates the priority of optimising monetary cost over time. In order to study the effectiveness and efficiency of the proposed heuristics, this chapter presents an extensive simulation study which analyses the best heuristic to adopt according to different user preferences.

5.1 Motivation

As discussed in the last chapter, users in the utility Grid may have conflicting requirements, and thus the coordination is needed with efficient and cost-effective scheduling algorithms. Other than this, users may also want to minimise application's response time not just monetary cost. In this scenario, the aim of user is to execute their applications most economically in the minimum time. The meta-broker needs to consider multiple factors such as execution time, monetary cost, and number of CPUs while making a scheduling decision on utility Grids where heterogeneous resources have different capabilities and pricing. Therefore, we designed three heuristics in order to simultaneously minimise the response time and monetary cost while meeting users' QoS requirements.

5.2 Meta-Broker System

As discussed in Chapter 3, resource providers sell the CPU time slots on their resources (clusters or supercomputers) and the consumers (or users) buy these time slots to run their

applications through the meta-broker. Providers delegate the allocation control of their CPUs to the meta-broker.

Users require their parallel applications with fixed CPU requirements to be executed in the most economical and efficient manner. Thus, the users also provide a trade-off factor to indicate the importance of cost over execution time, otherwise it will be set by the meta-broker. The trade-off factor can be calculated by a user on the basis of urgency and budget for executing the application.

The meta-broker maps the user's application to the delegated resources through advanced reservation based on the availability of CPUs and usage cost per second at regular time intervals. The objective of the meta-broker is to schedule all user applications such that both the total time and cost for applications execution are minimised.

5.3 Meta-Scheduling Algorithms

In general, users have two QoS requirements, i.e., the processing time and cost for executing their applications on pay-per-use services [176]. The users normally would like to get the execution done at the lowest possible monetary cost in minimum time. Thus, we introduce a trade-off factor that indicates the importance level of the monetary cost for users in comparison to the response time. In this section, we present our meta-scheduling heuristics that aim to manage the trade-off between cost and time.

5.3.1 Problem Statement

Let $n(t)$ be the number of user's resource request during the scheduling cycle which ends at time t. Every application i requires p_i CPUs for execution. Let $T(t)$ be the set of applications that the meta-broker has to schedule at time t. Let application i arrives at time $arr(i)$. The estimated time to compute (ETC) values of each application on each compute resource is assumed to be known based on the user-supplied information, experimental data, application profiling or benchmarking, or other techniques. We assume that an application cannot be executed until all of the required CPUs are available simultaneously. Let $m(t)$ be the total number of service providers available, and $R(t)$ be the set of service providers available during the scheduling interval at time t. Each service provider has m_i CPUs to rent. Let c_j be the cost of using a CPU on resource j per unit time.

Let $s(i,j)$ and $f(i,j)$ be the submission time and finish time of application i on resource j, respectively. The response time of application i is defined as

$$\alpha(i,j) = f(i,j) - s(i,j)$$

The average execution time of application i is given by

$$\beta_i = \frac{\sum_{j \epsilon R(t)} ETC(i,j)}{m(t)}$$

The cost spent to execute application i on resource j is given by

$$c(i,j) = c_j \times p_i \times ETC(i,j)$$

The average cost of executing application i is given by

$$\gamma_i = \frac{\sum_{j \epsilon R(t)} c(i,j)}{m(t)}$$

Given δ is the trade-off factor for all user applications; the trade-off cost metric for i_{th} user application is given by:

$$\phi(i,j,t) = \delta \frac{c(i,j)}{\gamma_i} + (1-\delta)\frac{\alpha(i,j)}{\beta_i} \tag{5.1}$$

Thus, the objective of our scheduling algorithm is to minimise the summation of trade-off metric for all user applications:

$$minimise(\sum_{\forall (i \epsilon T(t), t)} min_{\forall j,t} \phi(i,j,t))$$

The scheduling problem is to map every application $i \epsilon T(t)$ onto a suitable resource $j \epsilon R(t)$ to minimise the total execution time and monetary cost of all user applications.

A lower bound on the overall cost and makespan (or response time) of executing all applications successfully can be calculated as the minimum cost and makespan when value of the cost metric (Equation 5.1) is the lowest for all the applications. The value of the cost metric will be the lowest when (1) response time is minimal i.e. application's submission time is equal to its execution start time, and (2) application get scheduled on the cheapest and fastest resource. Let $\phi(i,j,arr(i))$ is the minimum for application i and resource j assuming no other application is present.
Mathematically, the lower bound can be described as follows :

$$Lower\ bound\ for\ Overall\ Cost = \sum_{\forall i} cost_{i,j}. \tag{5.2}$$

$$\begin{aligned} Lower\ bound\ for\ Overall\ Makespan &= max(max_{\forall i}(arr(i) + ETC(i,j)), \\ &\quad + \sum_{\forall i} ETC(i,j)/m) \end{aligned} \tag{5.3}$$

It is clear that the above lower bounds may not be achieved by the optimal schedule.

5.3.2 Min-Min Cost Time Trade-off (MinCTT) Heuristics

MinCTT is based on the concept of Min-Min heuristic [110][86][15]. For each user application, MinCTT finds a time slot on a resource with the minimum value of cost metric as defined in Equation 5.1. From these user application/time slot pairs, the pair that gives the overall minimum is selected; and that application is scheduled onto that time slot of the resource. This procedure is repeated until all of the user applications have been scheduled. The pseudo code for MinCTT is given in Algorithm 5.1. The meta-broker collects the user application's QoS requirements and available time slots from resources during schedule interval (line 1 and 2). MinCTT heuristic, then, selects the resource time slot for which value of the cost metric is minimum (line 3-8). From the feasible schedule queue, the heuristic assigns the time-slots to the user application with the minimum cost metric value (line 12-13). Then, the available time slots list is updated; and the process is repeated until all applications get the required resource time slot (line 14-16).

Algorithm 5.1: Pseudo code for MinCTT

Input: set of applications (submission time, execution time, CPUs required) and resources (time slots, number of available CPUs)
Output: Mapping of applications to resources

1 Collect all user applications until *Schedule Interval* ends
2 Get list of available time slots for all resources
3 **foreach** *user application* u_i **do**
4 **foreach** *each resource* r_j **do**
5 Find all feasible time slots
6 Find time slot *TS* which minimises cost metric $\phi(i,j,t) = \delta\frac{c(i,j)}{\gamma_i} + (1-\delta)\frac{\alpha(i,j)}{\beta_i}$
7 Insert *TS* and resource pair in feasible schedule queue *S*
8 **end**
9 $(TS_i, r_j) \leftarrow$ element with minimum cost metric value from S
10 Insert $(u_i, (TS_i, r_j))$ pair in a queue K
11 **end**
12 $(u, (TS, r)) \leftarrow$ element with minimum cost metric value from K
13 Allocate time slot TS on resource r to user application u
14 Update the time slots list for resource r
15 Remove u from user application list
16 Repeat $3 - 15$ until all applications are allocated

5.3.3 Max-Min Cost Time Trade-off (MaxCTT Heuristics)

MaxCTT is based on the concept of Min-Max heuristic [110][86][15]. This heuristic removes fragmentation from the time slot reservations. For each user application, first MaxCTT finds a time slot on a resource with the minimum value of cost metric as defined in Equation 5.1. From these user application/time slot pairs, the pair that gives the overall

maximum is selected; and that application is scheduled onto that resource's time slot. This procedure is repeated until all of the user applications are scheduled. The pseudo code to MaxCTT is the same as for MinCTT, except replace "minimum" with "maximum" in line 12 of Algorithm 5.1. MaxCTT heuristic, then, selects the resource time slot for which value of the cost metric is minimum (line 3-8). From, the feasible schedule queue, the heuristic assigns the time-slot to the user application having the maximum cost metric value (line 12-13). Then, the available time slots list is updated and the process is repeated until all applications get the required resource time slot (line 14-16).

5.3.4 Sufferage Cost Time Tradeoff (SuffCTT Heuristics)

SuffCTT is based on the concept of Sufferage heuristic [110][116] which assigns highest priority to the application, which would "suffer" the most if not assigned. For each user application, first SuffCTT finds the time slot on a resource with minimum sufferage value which is the difference between its best and second best value of cost metric as defined in Equation 5.1. From these user application/time slot pairs, the pair that gives the highest sufferage value is selected; and that application is scheduled onto that time slot of that resource. This procedure is repeated until all of the user applications have been scheduled. The pseudo code for SuffCTT is given by Algorithm 5.2. SuffCTT heuristic, then, selects for each application, all the feasible time-slots for which value of the cost metric is minimum (line 3-8). The heuristic calculates the sufferage value for each application and select the application with the maximum sufferage value (line 9-13). The selected application is then scheduled on the resource time slot with the minimum cost metric value (line 14). Then, the available time slots list is updated and the process is repeated until all applications get the required resource time slot (line 15-17).

5.3.5 Time Complexity

Let $TS(j,t)$ be the number of time slots initially available at resource j during the scheduling interval at time t. The main operations performed during MinCTT and MaxCTT are the following:

1. To allocate any resource to an application, the worst number of iteration is to be performed over each user application and resource i.e. $O(n * |R(j)|)$ times

2. In each iteration (step 5 to 8 in Algorithm 5.1), time slot with minimum execution time is searched. This is of order of available time slots i.e., $O(TS(j,t))$.

3. In worst case, the above operations are performed for each application, i.e., n times

Algorithm 5.2: Psuedo code for SuffCTT

Input: set of applications (submission time, execution time, CPUs required) and resources (time slots, number of available CPUs)
Output: Mapping of appications to resources

1 Collect all user applications until *Schedule Interval* ends
2 Get list of available time slots for all resources
3 **foreach** *user application* u_i **do**
4 **foreach** *each resource* r_j **do**
5 Find all feasible time slots
6 Find time slot *TS* which minimises cost metric $\phi(i,j,t) = \delta\frac{c(i,j)}{\gamma_i} + (1-\delta)\frac{\alpha(i,j)}{\beta_i}$
7 Insert *TS* and resource pair in feasible schedule queue *S*
8 **end**
9 Calculate the sufferage s_i value for application u_i
10 $(TS_i, r_j, s_i) \leftarrow$ element with minimum cost metric value from S
11 Insert $(u_i, (TS_i, r_j), s_i)$ pair in a queue K
12 **end**
13 $(u, (TS, r), s_i) \leftarrow$ element with maximum sufferage value from K
14 Allocate time slot TS on resource r to user application u
15 Update the time slots list for resource r
16 Remove u from user application list
17 Repeat $3 - 15$ until all applications are allocated

Therefore, the resultant worst case complexity of the meta-scheduling algorithm is a combination of above operations, i.e., $O(n^2 \sum_{j \epsilon R(t)} TS(j,i,t))$. In the case of SuffCTT, the worst time complexity will be similar as the number of operations in SuffCTT and other other proposed heuristics are almost same.

5.4 Simulation Setup

User applications are modelled as parallel applications which require all CPUs to be allocated at the same time and on the same resource. About 1,000 user applications are generated according to the Lublin workload model [109]. The model specifies the arrival time, number of CPUs required, and execution time (μ) of application. This model is derived from existing workload traces for rigid jobs and incorporates correlations between job runtimes, job sizes, and daytime cycles in job inter-arrival times. The workload parameters values we used for Lublin model are listed in Table 5.1. Since the generated workload gives execution time on one resource, the ETC matrix is thus generated using random distributions to simulate the effect of heterogeneous resources. The variation of the application's execution time on different resources can be high or low. A high variation in execution time of the same application is generated using the gamma distribution method presented by Ali et al. [2]. In the gamma distribution method [2], a mean task execution time and coefficient of variation (COV) are used to generate ETC matrices. The mean task execution time of an application is set to μ and a COV value of 0.9 is used.

Similarly, the low variation in the execution time is generated using uniform distribution with mean value of μ and standard deviation of 20 seconds.

Table 5.1: Lublin Workload Model Parameter Values

Workload Parameter	**Value**
jobType	$BATCH\_JOBS$
Maximum number of CPUs required by a job (p)	1000
uHi	$log_2(p)$
$uMed$	$uHi - 2.5$
Other parameters	as created by Lublin model

Table 5.2: Simulated EDG Testbed Resources.

Site name (Location)	**Number of CPUs**	**Single CPU rating (MIPS)**	**Execution price (G$)**
RAL (UK)	20	1140	0.0061
Imperial College (UK)	26	1330	0.1799
NorduGrid (Norway)	265	1176	0.0627
NIKHEF (Netherlands)	54	1166	0.0353
Lyon (France)	60	1320	0.1424
Milano (Italy)	135	1000	0.0024
Torina (Italy)	200	1330	1.856
Catania (Italy)	252	1200	0.1267
Padova (Italy)	65	1000	0.0032
Bologna (Italy)	100	1140	0.0069

The computing installation modelled in our simulation is that of a subset of the European Data Grid (EDG) 1 testbed [83] which contains ten Grid resources spread across four countries connected via high capacity network links. The configurations assigned to the resources in the testbed for the simulation are listed in Table 5.2. The configuration of each resource is decided so that the modelled testbed would reflect the heterogeneity of platforms and capabilities. All the resources were simulated as clusters of CPUs that employed easy backfilling policies and allow advance reservation in order to improve responsiveness. The number of CPUs on each resource are chosen such that the demand of CPUs by all applications will always be greater than the total free CPUs available on all the resources. A sample of initial price of using each PE on a Grid resource is given in Table 5.2. In the real world, pricing of resources are generally controlled by many economic factors. As our research focus is not on how to price the resources and also for simplicity, we generated the prices randomly using distributions. The prices of resources are generated using Weibull Distribution with parameters $\alpha = 0.4$ & $\beta = 0.8$. The pric-

ing of resource may or may not be related to CPU speed. Thus, minimisation of both execution time and cost of an application may conflict each other depending on how the resources are priced. Thus, two types of prices, i.e., consistent and inconsistent prices, for resources are used in the experiments. The consistent prices of resources means the prices of resources are inversely proportional to their CPU speed. Thus, the price of the slowest resource will be the lowest. Otherwise, the pricing of resources will be referred to as inconsistent. These resources send the availability of time slots to the meta-broker regularly. The schedule interval of the meta-broker is 50 simulation seconds.

We compare our proposed heuristics (denoted as MinCTT, SuffCTT and MaxCTT) with a common heuristic which is used in the previous work i.e. cost-based Greedy heuristic (Greedy). This approach is derived from the cost optimisation algorithm in Nimrod-G [1], which is initially designed for scheduling independent tasks on Grids and thus enhanced for parallel applications. The enhanced Greedy heuristic allocate resources to applications on first-come-first-serve basis. To schedule each application, it first calculates the cost metrics (Equation 5.1) for all the time slots available on each of the resources and, then schedules the application to the time slots with the minimum trade-off cost.

As discussed in previous sections, the trade-off factor is used to manage the users preference for the allocation and the execution cost. The trade-off factor can be decided by one of the two participants i.e., users or meta-broker. Thus, either each user can submit to the meta-broker their trade-off factor or the meta-broker can set one trade-off factor for all applications on behalf of the users.

Hence, the experiments are conducted for the following **two cases:**

1. **Case 1:** the trade-off factor is set by the meta-broker.
2. **Case 2:** the trade-off factor is provided by each user.

with **four configurations:**

1. High variation in execution time and inconsistent prices of resources (HIUC)
2. High variation in execution time and consistent prices of resources (HICC)
3. Low variation in execution time and inconsistent prices of resources (LOUC)
4. Low variation in execution time and consistent prices of resources (LOCC)

The two metrics used to evaluate the scheduling approaches are overall makespan and average execution cost. The former indicates the maximum time when all the submitted applications finish execution, whereas the latter indicates how much it costs to schedule all the applications on the testbed.

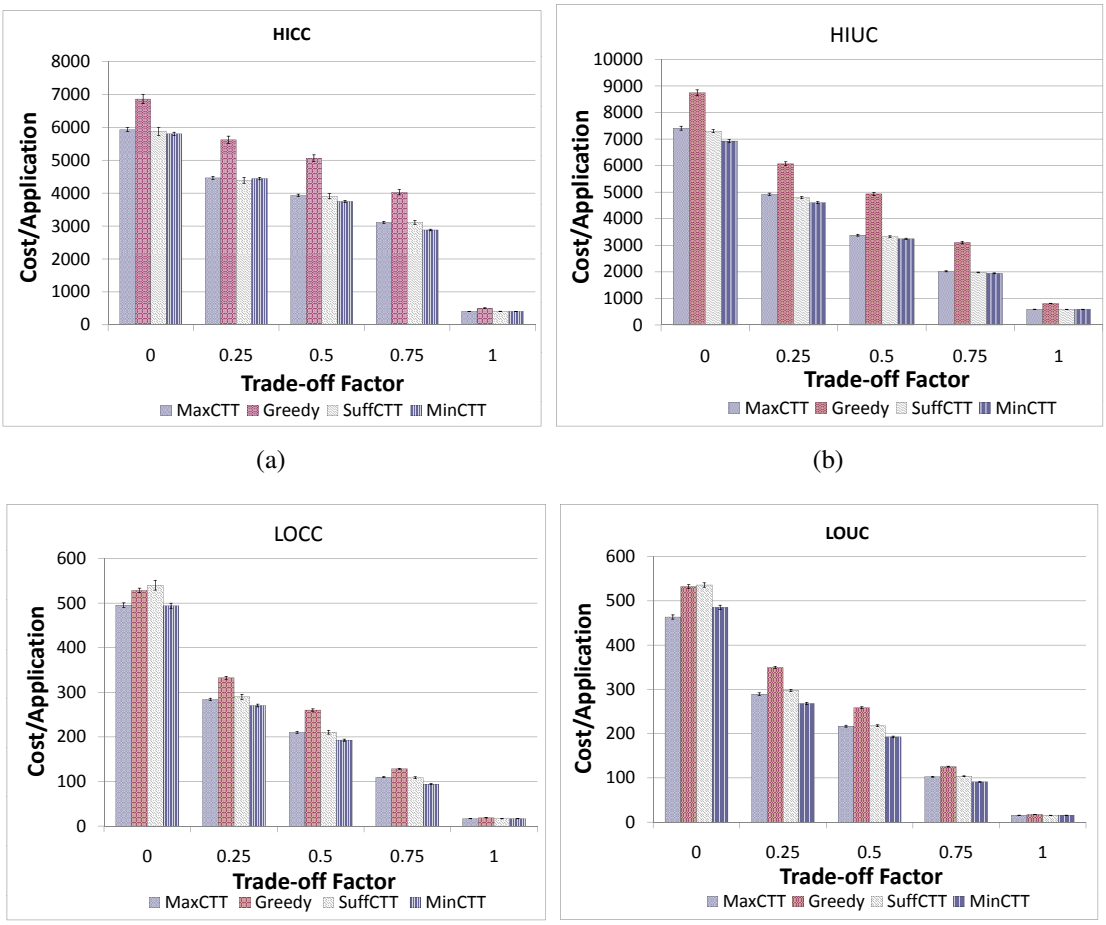


Figure 5.1: Overall Average Cost of Execution

5.5 Analysis of Results

This section shows the comparison between MaxCTT, Greedy, SuffCTT, and MinCTT heuristics. This section also shows how the proposed heuristics reduces combined execution cost and makespan of applications submitted during all scheduling intervals.

5.5.1 CASE 1: Trade-off Factor Set by Meta-broker

The results (with 95% confidence interval bars) for various configurations with varying trade-off factor by the meta-broker are shown in Figure 5.1 and 5.2. The results presented are averaged out over ten trials with different resource prices. This section presents the effect of different trade-off factors on the performance of heuristics. Greedy heuristic performed worst in minimising the overall execution cost and makespan. For HICC and HIUC configurations (Figure 5.1(a) and 5.1(b)), Greedy heuristic results in about 15% more average execution cost than other heuristics. Similarly, for LOCC and LOUC con-

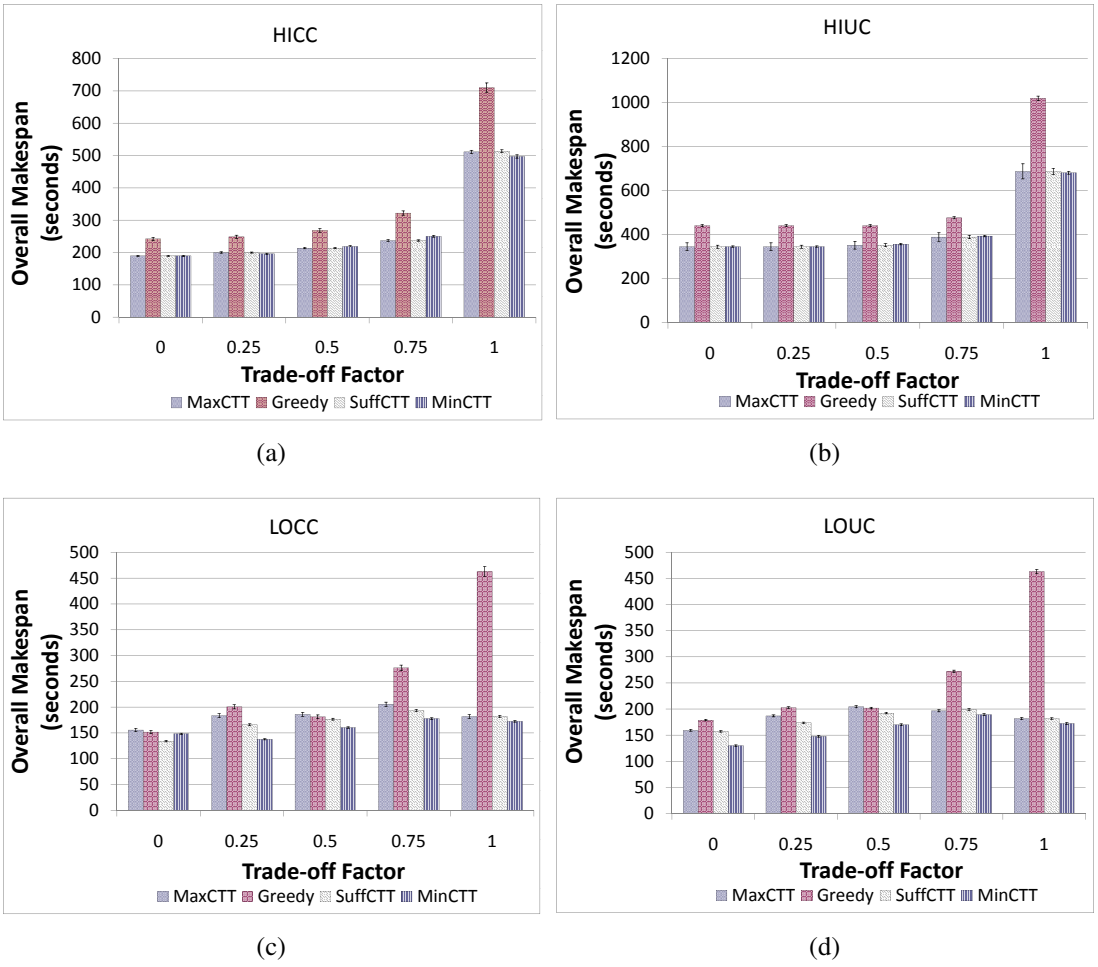


Figure 5.2: Overall Makespan

figuration (Figure 5.1(c) and 5.1(d)), Greedy heuristic also results in about 10% more execution cost except when "Trade-off factor=0". The reason for this anomaly is that "Trade-off factor=0" means users are only looking for the fastest execution time of their applications, not the cheapest resource. Thus all heuristics try to run the applications on the fastest resource. For HICC and HIUC configurations (Figure 5.1), MaxCTT, MinCTT and SuffCTT results in almost similar average execution cost. While in the case of LOCC and LOUC configurations, MinCTT resulted in the lowest cost.

It can be noted from Figure 5.1 that with the increase in the trade-off factors, the overall average execution cost is also decreased. With the increase in the trade-off factor from 0 to 1, the average execution cost fall by about 90%. This is because more applications are scheduled on the cheaper resources, due to the increase in user's priority for cost over time.

Figure 5.2 shows the effect on the overall makespan by the four heuristics in different configurations. The overall makespan of applications for HICC and HIUC (Figure 5.2(a) and 5.2(b)) is about 30% higher than for LOCC and LOUC (Figure 5.2(c) and 5.2(d)).

Moreover, for LOCC (Figure 5.2(c)) and LOUC (Figure 5.2(d)), the overall makespan by all heuristics, except greedy, remained almost the same (about 150–200 sec) regardless of the change in trade-off factor (0–1). This is because the low variation in execution time across various resources causes the time factor in the cost metric (as defined in Equation 5.1) to be less effective. This low variation in overall makespan also demonstrates the effectiveness of our proposed heuristics in managing the time requirement of all users.

For configurations HICC and HIUC (Figure 5.2(a) and 5.2(b)), the overall makespan increases with the trade-off factor which becomes very significant for Trade-off factor=1. This increase in the overall makespan is because of running all applications on the cheapest resources. Figure 5.2 shows that MinCTT resulted in the lowest overall makespan and greedy resulted in the highest makespan.

The effect of cost consistency can also be observed when we compare Figure 5.1 and 5.2. The overall makespan is increased on average by about 30–40% from HICC configuration (Figure 5.2(a)) to HIUC configuration (Figure 5.2(b)). Similar behaviour is also observed in the case of average execution cost from Figure 5.1(a) and 5.1(b).

5.5.2 CASE 2: Trade-off Factor Set by User

This section discusses the performance of the heuristics in four different configurations of ETC matrix and resource pricing.

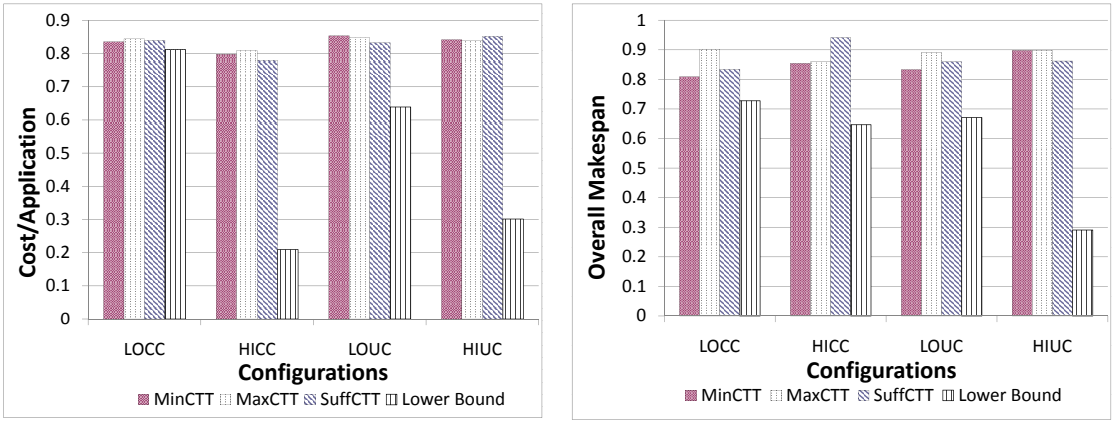
Impact on Makespan and Execution Cost of Users

In Figure 5.3, the normalised average execution cost and overall makespan for all user applications is compiled in four different configurations with the lower bound. The cost and makespan by different heuristics is normalised using values by Greedy heuristic.

The Greedy heuristic performed worst by generating the most expensive schedule with the maximum makespan in almost all four configurations, thus values in Figure 5.3 are less than one. MaxCTT, MinCTT, and SuffCTT resulted in about 18% cheaper schedule than Greedy heuristic. The reason for Greedy heuristic performance is that Greedy heuristic does not consider the effect of other applications in the meta-broker while generating the schedule for any application. Even though, MinCTT, MaxCTT, and SuffCTT heuristics give almost same overall cost improvement over Greedy, MinCTT gives the best overall makespan in almost all the cases. For example, in LOCC configuration (Figure 5.3(b)), MinCTT improved the overall makespan by 20%.

Moreover, in comparison to the lower bound, our proposed heuristics give the best performance for LOCC and LOUC configuration. In Figure 5.3, in case of LOCC configuration, the difference between lower bound for overall cost and our heuristics is just $2 - 3\%$, while for overall makespan it is about $8 - 9\%$. In case of high variation of

execution time (i.e. for HICC and HIUC configuration), the difference is large.



(a) Total Execution Cost of Applications

(b) Total Makespan of Applications

Figure 5.3: Different ETC and Resource Pricing Configurations

Application Distribution on Resources

Figure 5.4 shows how the applications are distributed on various resources by the meta-scheduling heuristics in four different configurations. This measure is taken to study how the pricing of resources affects the selection process of the heuristics. In Figure 5.4, it can be observed that, a maximum number of applications are allocated on Grid sites such as NorduGrid and Catania in all the configurations i.e. LOUC, HIUC, LOCC and HICC. For example, in Figure 5.4(b), about 25% applications are scheduled on Catania. This is due to the fact that NorduGrid and Catania have the maximum number of CPUs thus more applications can be scheduled which results in lower makespan.

When cost is consistent with the ETC values of application, less variation in execution time of applications across resources results in the assignment of more applications to the cheapest resource as the effect of execution time is very low. Thus, it can be noted in Figure 5.4(a)-5.4(d) that for HICC and LOCC configurations, the number of applications on Catania has decreased, but has increased on Bologna and NorduGrid.

For LOCC and HICC configurations, we also observe that on the Toriana resources more applications are scheduled even though it has highest price (Table 5.2) than RAL resources. This is due to two reasons, first, RAL has only 20 CPUs, thus it can run fewer applications than Imperial College. Second, even though Imperial College is expensive, it is the fastest resource thus it decreases in the weight of time factor ($\alpha(i, j)$) in the cost metric as defined in Equation 5.1.

From Figure 5.4, the reason for lower total execution cost in case of MaxCTT, MinCTT, and SuffCTT is also clear. MaxCTT, MinCTT, and SuffCTT have allocated more applications on cheaper resources than the Greedy heuristics.

The effect of cost consistency is very low when the variation in execution time of an application is less across different resources. It can be observed in Figure 5.4(a)-5.4(d) that the distribution of application in LOUC and LOCC configurations is similar. For example, in Figure 5.4(b) on NorduGrid about 15–17% applications are scheduled in both LOCC and LOUC configurations. However, in the case of high variation in execution time of applications across the resources, the effect of cost consistency is quite high. For example, in Figure 5.4(b), the percentage of applications scheduled by MaxCTT on Bologna resources is about 2% in HICC configuration; which increases to about 12% in case of HIUC configuration. A similar pattern of application distribution can be observed in Figure 5.4(b)–5.4(d). The reason for this behaviour is the trade-off between execution time and cost. For any application, all the heuristics have to choose a resource which is not only faster but also cheaper.

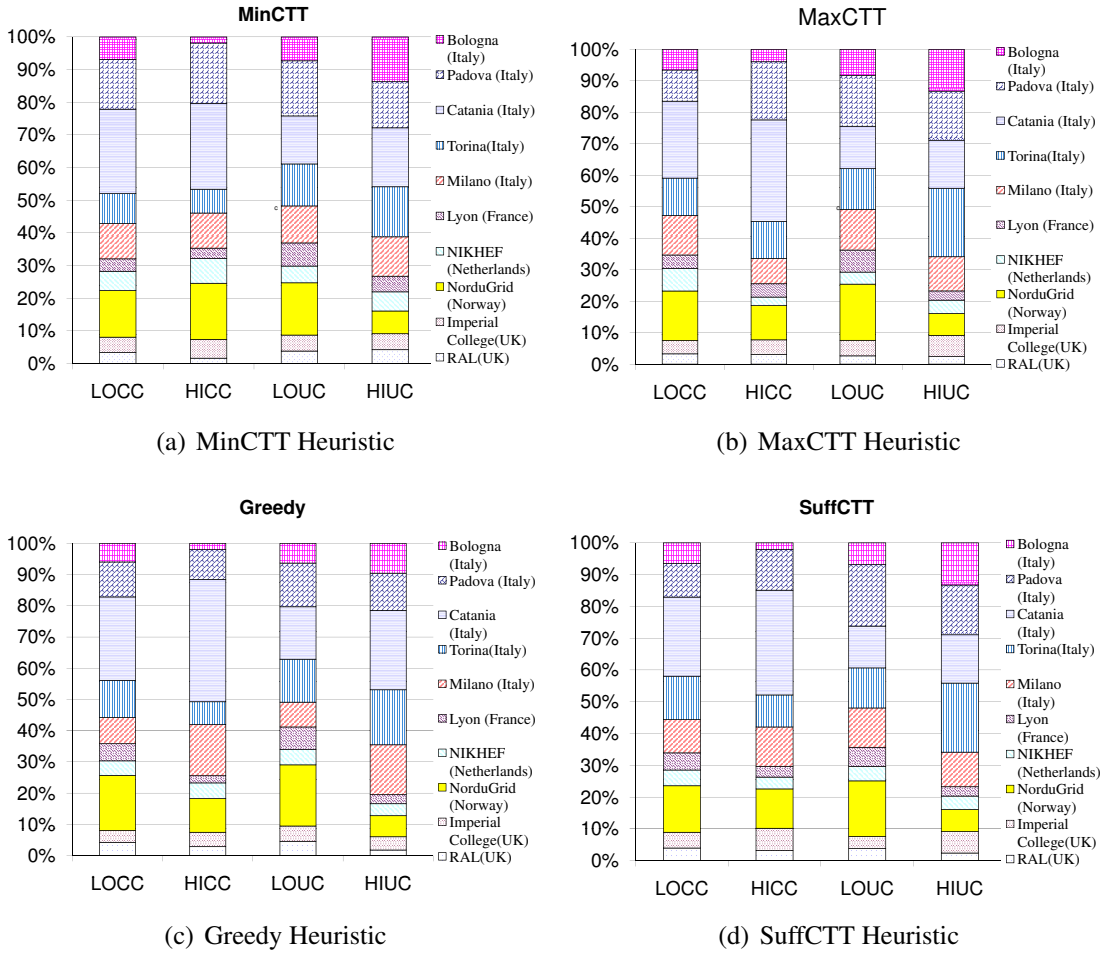
Effect of Scheduling Interval

In this experiment, the performance of heuristics is analysed with varying scheduling interval. We increased the submitted applications to 1500 and number of resources to 15, where number of processors were randomly generated (between 20–300) by uniform distribution. The results are presented for HICC configuration in Figure 5.5. Other configuration results are not presented to save space as they are very similar. Figure 5.5 clearly shows that Greedy heuristic performed worst with the highest cost and makespan which is about 25% higher than other heuristics and remained almost constant across all scheduling intervals. The average cost (Figure 5.5(a)) of all heuristics is unaffected by the changes in the scheduling interval. The overall makespan (Figure 5.5(b)) changed drastically for MaxCTT and SuffCTT where it is dropped by 15% and increased by 15% respectively.

Effect of Input and Output Transfer Cost

This experiment is conducted to analyse the effect of input and output data transfer cost on the performance of heuristics. We have considered data transfer time associated with each application. Since applications are compute-intensive, the data transfer cost is modelled to be 1/10th of the average execution cost of application. To vary data transfer cost over time i.e. to simulate changes in available bandwidth with time, we have considered a bandwidth factor which is submitted by resource provider to the meta-broker in every scheduling interval. The bandwidth factor is randomly generated (between 0–1) using uniform distribution. In each scheduling interval, data transfer cost (DTC) of an application to execute on any resource r is given by:

$$DTC(t) = \frac{Intial\_DTC}{bandwidth\_factor(r,t)}$$



(a) MinCTT Heuristic

(b) MaxCTT Heuristic

(c) Greedy Heuristic

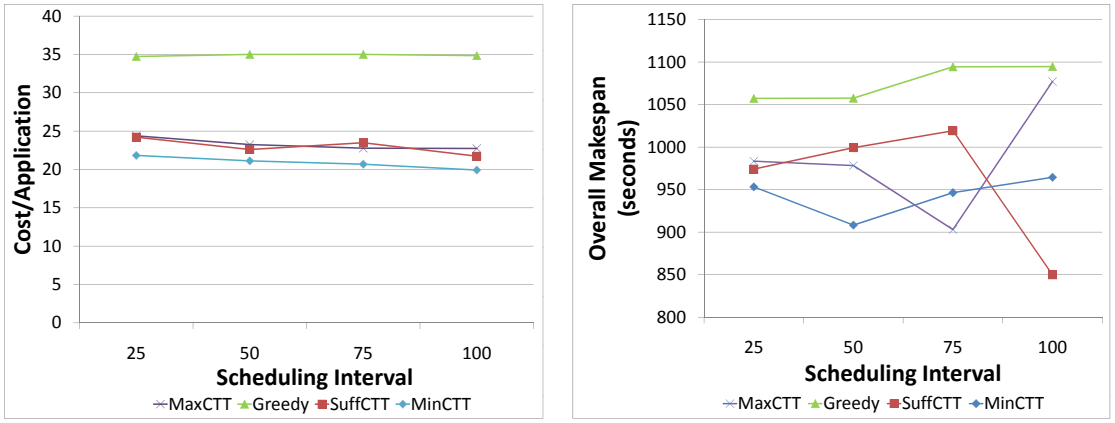
(d) SuffCTT Heuristic

Figure 5.4: User Application Distribution on Resources in Different Configurations

The data transfer cost will be added in the total execution cost of the application. Figure 5.6 shows the average cost and overall makespan in different configurations with DTC. Clearly Greedy heuristic generates the most expensive schedule in all four configurations. Even though the makespan seems to be improved by Greedy heuristic for LOCC and LOUC configurations, due to the trade-off factor, the execution cost by Greedy heuristic is about 10–25% higher than other heuristics. MinCTT resulted in the best overall makespan almost in all configurations. If we compare results in Figure 5.3 when DTC is not considered and Figure 5.6 when DTC is considered, it is clear that the average cost resulted by Greedy is still most expensive, while the overall makespan is improved when the variation in execution time of application across resources is low.

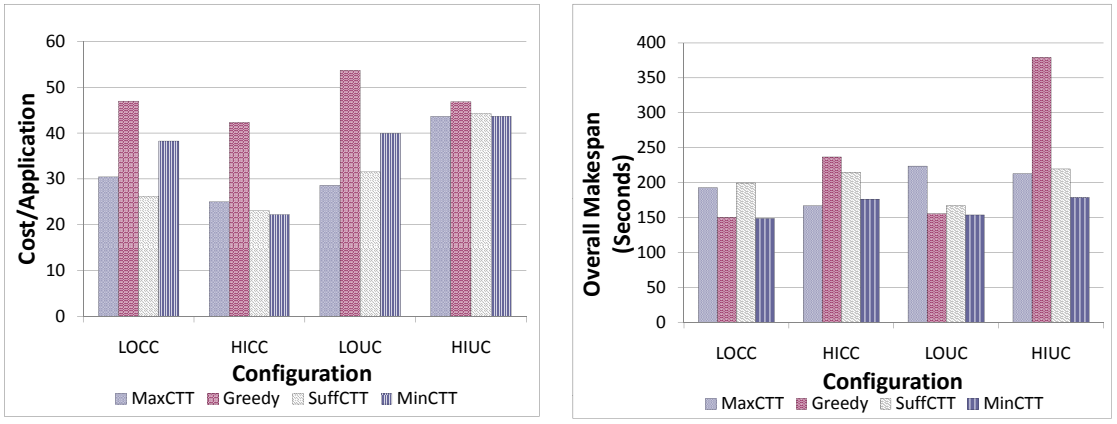
Effect of Different Job Sample Size

In previous experiments, the number of applications submitted during experiments were fixed. Here, we vary the number of applications and analysed how average cost and overall



(a) Total Execution Cost of Applications (b) Total Makespan of Applications

Figure 5.5: Effect of Scheduling Interval in HICC Configuration



(a) Total Execution Cost of Applications (b) Total Makespan of Applications

Figure 5.6: Effect of DTC on Cost and Time

makespan changes in different configurations. The results are presented in Figure 5.7 and 5.8. We also consider the effect of DTC for this experiment. Figure 5.7 and 5.8 show that the average execution cost and overall makespan increase with the number of submitted applications. Greedy heuristic resulted in a schedule which is as expensive as 70% (Figure 5.7) of schedule generated by other heuristics. For LOUC and LOCC configurations (Figure 5.8(c) and 5.8(d)), as variation in execution time is low, due to which the time effect on scheduling decision is negligible. Thus, even though Greedy heuristics seems to outperform other heuristics in these configurations (Figure 5.8(c) and 5.8(d)), it results in very costly schedule as shown in Figure 5.7(c)– 5.7(d). This becomes more clear from results in Figure 5.8(a)– 5.8(b) for HICC and HIUC configurations.

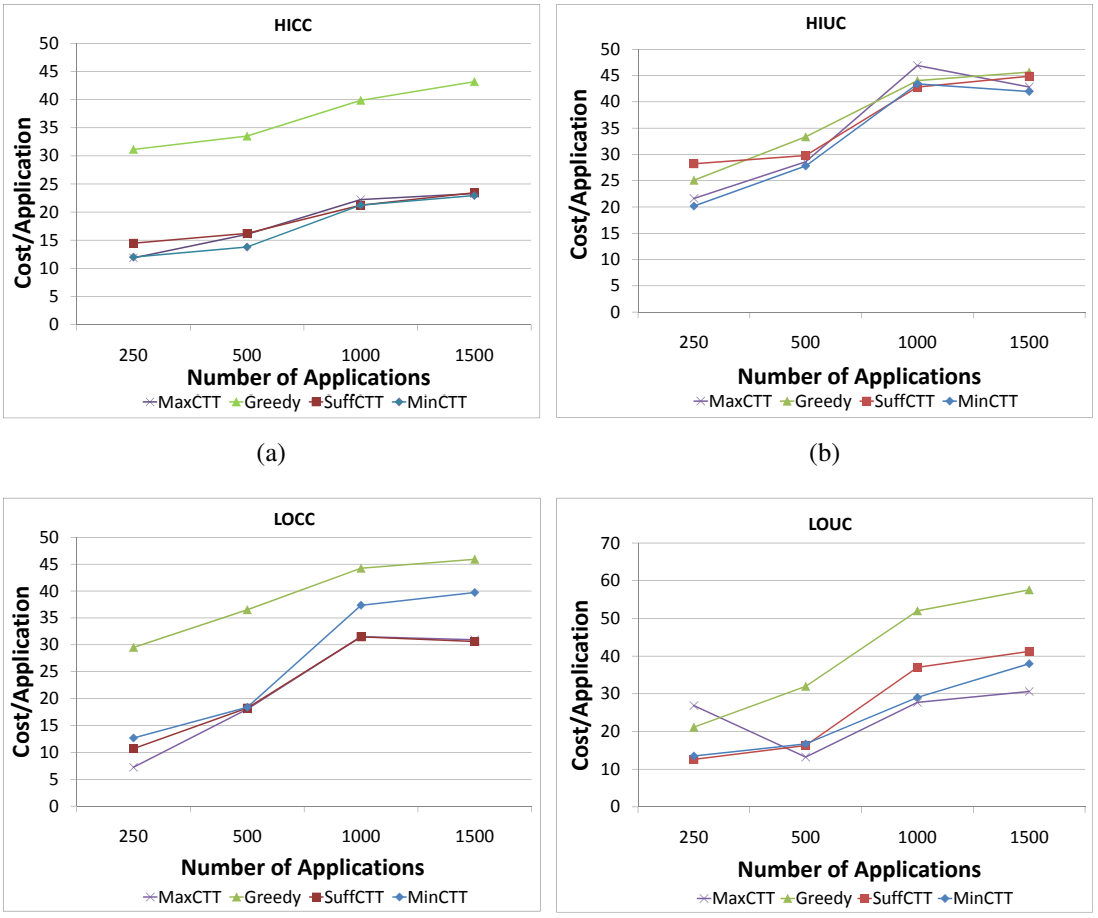


Figure 5.7: Effect of Change in Application Submitted on Cost

5.6 Related Work

As noted in Chapter 2, many projects have investigated the application of market models and mechanisms to improve the efficiency of scheduling in Grids. In this section, we discuss and compare the most relevant works in the context of minimising both response time and cost for Grid users.

Feng et al. [61] proposed a deadline cost optimisation model for scheduling one application with dependent tasks. This work is different from our research in two ways: (1) algorithms proposed are not designed to accommodate concurrent users competing for resources; (2) the application model considered is for applications whose tasks can be distributed on different resource sites. Munir et al. [116] proposed QoS Sufferage for independent task scheduling on grids. This work does not consider the cost and time trade-off and only focused on improving makespan. Kumar et al. [101] proposed two heuristics, HRED and HRED-T, to minimise business value, i.e., cost or time, of users. In this work, they studied the minimisation of only one parameter, i.e., cost, but not mul-

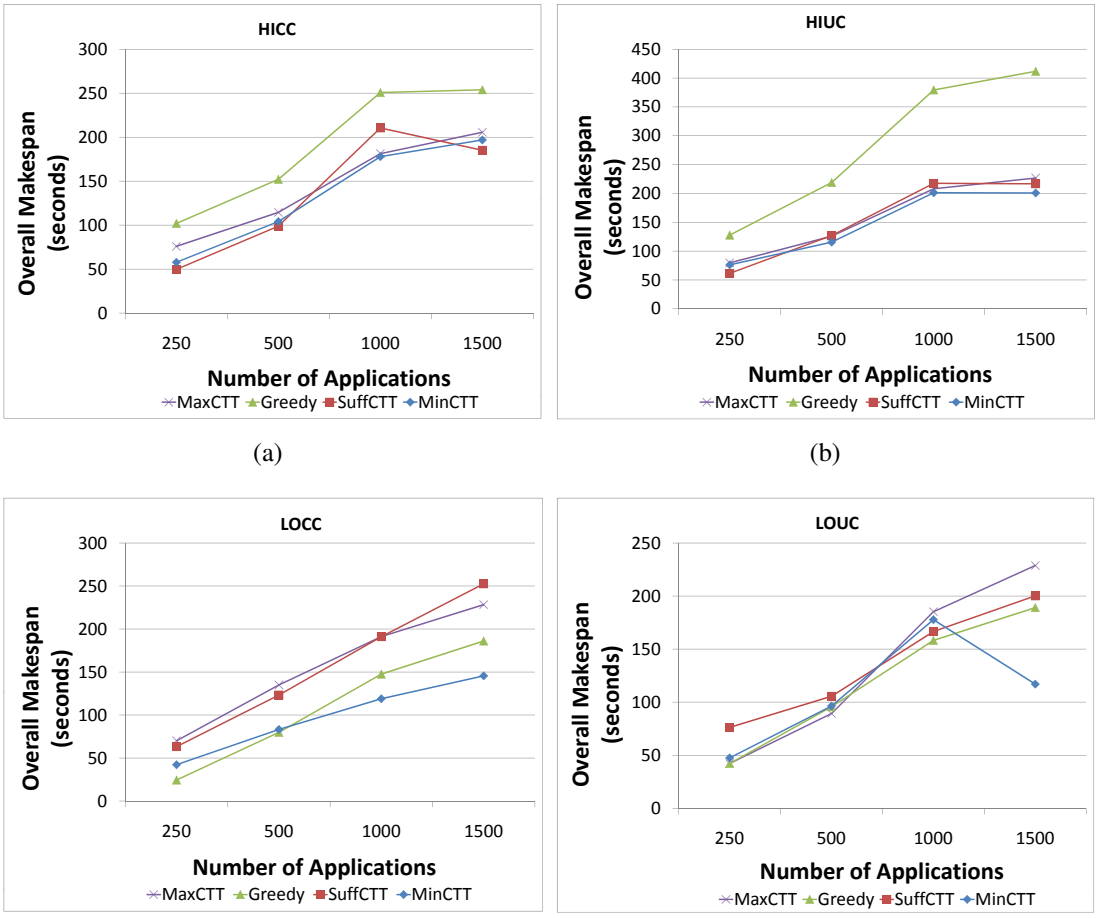


Figure 5.8: Effect of Change in Application Submitted on Overall Makespan

tiple conflicting parameters such as time and number of processors required. Dogan et al. [46] proposed a meta-scheduling algorithm considering many concurrent users, but the application model assumed that each application consists of one task that requires only one processor and each application is independent. In this chapter, we have considered multiple and concurrent users competing for resources in a meta-scheduling environment to minimise the trade-off between the combined cost and time of all user applications.

Many Genetic Algorithms (GA)-based heuristics are also proposed in the literature. Kim et al. [98] proposed a novel GA-based algorithm which schedules a divisible data intensive application. Martino et al. [45] presented a GA-based scheduling algorithm where the goal of super-scheduling was to minimise the release time of jobs. Wang et al. [166] considered the use of a GA to schedule a DAG of communicating tasks onto a heterogeneous parallel system to minimise makespan. These GA-based heuristics based solutions do not consider QoS constraints of concurrent users such as budget and deadline. Singh et al. [147] presented a multi-objective GA formulation for provisioning resources for an application using a slot-based resource model to optimise cost and performance.

As GAs require a long time to execute, they are not suitable for a dynamic environment such as Grids where schedules have to be recomputed regularly as resource availability changes rapidly.

For scheduling approaches outside Grid computing, the Min-Min, Min-Max, and Sufferage heuristics [110] are the three major task-level heuristics employed for resource allocation. As they are developed based on specific domain knowledge, they cannot be applied directly to scheduling parallel jobs with strict requirements on utility Grids. In this chapter, we are considering jobs/applications with fixed number of processors requirement that are also called rigid parallel jobs [145]. The scheduling of such jobs on Grid resources is a complex 0–1 Knapsack Problem that is more challenging than traditional scheduling on parallel systems, because of the fixed (rigid) number of processors required by the job, the dynamic availability of resources with different capabilities in different administrative domains, and continuously arriving jobs at the meta-scheduler [178]. Thus, in this chapter, these heuristics are enhanced accordingly.

5.7 Summary

In utility Grids, a user may have conflicting goals of minimising simultaneously the processing time and monetary execution cost of his/her application. Thus, we design a cost metric to manage the trade-off between execution cost and time. We also propose three meta-scheduling heuristics (MaxCTT, SuffCTT, and MinCTT) which minimise and manage the execution cost and makespan of users' applications.

The sensitivity of the proposed heuristics is evaluated with respect to changes in the user preferences (trade-off factor), application's execution time, and resource pricing. The results show that MaxCTT, SuffCTT, and MinCTT outperform the Greedy heuristic in not only optimising cost, but also in minimising the overall makespan. When the trade-off factor (TF) value is chosen by the meta-broker, MinCTT gives the lowest makespan and cost for all configurations except for $TF = 1$. When the trade-off factor is set by users, MinCTT, SuffCTT, and MaxCTT generate the cheapest schedule with the lowest makespan. Among these three heuristics, MinCTT results in the lowest overall makespan for LOCC, LOUC and HICC configurations. For HIUC configuration, SuffCTT generates the lowest overall makespan. The impact on the average cost due to the changes in the scheduling interval is minimal for all heuristics. Similarly, the overall makespan by MaxCTT and Greedy heuristics almost remained same. When data transfer cost is considered, similar results are observed except for LOUC and LOCC configurations, MinCTT and Greedy heuristic generates the lowest overall makespan. In short, this chapter has addressed the importance of managing the trade-off between execution cost and response time by analysing various resource configurations.

Chapter 4 and 5 discusses the meta-scheduling of concurrent users with competing QoS requirements. The meta-broker maximises users' utility by decreasing their spending for execution of applications in the utility Grid environment. In the next chapter, we will discuss the market-oriented meta-scheduling problem in the second scenario, i.e. maximising the provider's utility (profit) while reconciling users' demand for resources.

Chapter 6

Meta-Scheduling to Maximise Provider's Utility

In this chapter, we study the problem of scheduling concurrent users with deadline constraint on multiple resource sites from the resource provider's perspective. In this context, the aim of our meta-broker is to maximise resource provider's utility calculated in terms of the total profit. The profit of resource provider can be maximised by two methods one by intelligent selection of users, and other by intelligent scheduling across different resource sites. In this chapter, we focus on minimising the resource provider's cost by using the later approach i.e. scheduling applications across multiple sites.

6.1 Motivation

The increasing maintenance cost of super-computing centers or data centers has become a big issue for resource providers. The major contributor in this cost is the high energy consumption [12] which checks the growth of provider's profit with growing resource demand [133]. The high energy consumption not only translates into high energy costs which directly affect the profit, but also high carbon emissions [113] which can indirectly affects the profit due to restriction from regulatory agencies [96]. Thus, many resource providers are building different data centers and deploying them in many geographical locations so as not only to expose their resources to business and consumer applications, e.g. Amazon [6], but also to reduce energy cost, e.g. Google [111]. To address this issue, in this chapter, we present novel near-optimal meta-scheduling algorithms to maximise provider's profit by exploiting heterogeneity across multiple resource sites in terms of energy efficiency and electricity cost.

The analysis of previous work shows that little investigation has been done for both economic and environmental sustainability of utility Grids. Most previous works have

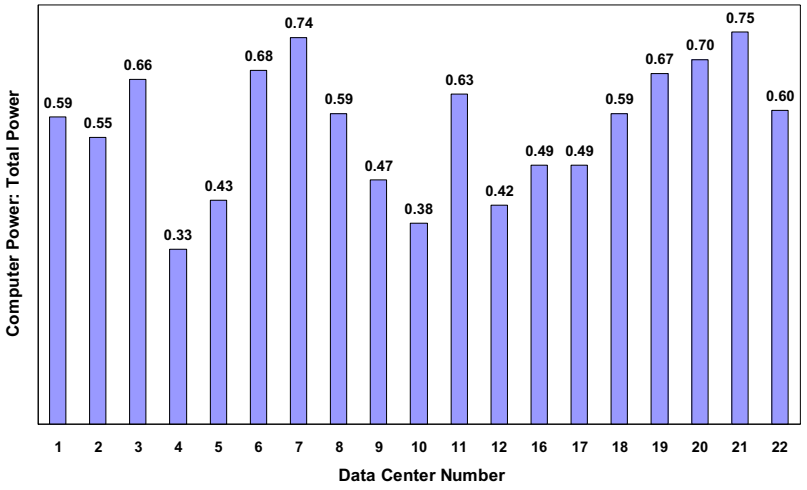


Figure 6.1: Computer Power Consumption Index (Source: [73])

studied how to reduce energy usage from the perspective of reducing cost, but not how to improve the profit while reducing the carbon emissions which are also significantly impacting the resource providers [62]. For example, to address energy usage, Chase et al. [32] adopted an economic approach to manage shared server resources in which services "bid" for resources as a function of delivered performance. Similarly, Burge et al. [19] scheduled tasks to heterogeneous machines, and made admission decisions based on the energy costs of each machine to maximise the profit of a single data center. But, both of them do not study the critical relationship between carbon emissions (environmental sustainability) and profit (economic sustainability) for the energy sustainability issue, and how they can affect each other.

Other previous works have focused on achieving energy efficiency at a *single* data center/supercomputing location, but not across *multiple* locations. However, resource providers such as Amazon EC2 [6] typically have multiple data centers distributed worldwide. As shown in Figure 6.1, the energy efficiency of an individual data center in different locations changes dynamically at various times depending on a number of factors such as energy cost, carbon emission rate, workload, CPU power efficiency, cooling system, and environmental temperature. Thus, these different contributing factors can be considered to exploit the heterogeneity across multiple data centers for improving the overall energy efficiency of the resource provider.

Moreover, several works have mainly proposed energy-saving policies that are application specific [63][67], processor-specific [137][50], and/or server-specific [167][97]. But, these policies are only applicable or most effective for the specific models that they are specially designed for. Hence, we require some simple, yet effective generic energy-efficient scheduling policies, that can be extended to any application, processor, and server models, so that they can be readily deployed in existing data centers with minimum changes at the infrastructure level. Table 6.1 summarises the most relevant previous

Table 6.1: Comparison of Related Work

	CO_2 emission / energy consumption	HPC workload characteristic	Multiple data centers	Energy cost aware scheduling	Market-oriented schedulers
Our work	X	X	X	X	X
Bradley et al. [14]	X	X			
Lawson and Smirni [105]	X	X			
Tesauro [158]	X	X			
Orgerie et al. [126]	X	X	X		
Patel et al. [129]			X		
Chase et al. [32]				X	X
Burge et al. [19]	X	X		X	

work which addresses any of the five aspects considered in this chapter.

Therefore, in this chapter, our aim is to design generic scheduling policies that can maximise the profit provider on the global scale; while they can easily complement any of these application-specific, processor-specific, and/or server-specific energy-saving policies that are already in place within existing data centers or servers.

6.2 Meta-scheduling Model

In this chapter, the system model considered is similar to previous chapters except the objective of the meta-broker is now to maximise the profit of resource provider while minimising application's deadline violations and carbon emissions. The meta-broker interprets and analyses the service requirements of a submitted application, and decides whether to accept or reject the application based on the availability of CPUs.

Users submit parallel applications with their QoS and processing requirements. Each application must be executed within an individual data center and does not have preemptive priority. Furthermore, since the main aim of work presented in this chapter is to design high-level application-independent meta-scheduling policies, we do not want to consider the fine-grained details of workloads (such as considering the impact of communication and synchronisation, and their overlapping with computation), which are more applicable at the local scheduler level. The objective of the user is to have his/her application com-

pleted by the specified deadline. Deadlines are hard, i.e. the user will benefit from the resources only if the application completes before its deadline [131].

A resource provider has multiple resource sites distributed across the world. Each resource site is modelled as a data center. For example, Amazon [6] has data centers in many cities across Asia, Europe, and United States. Each data center has a local scheduler that manages the execution of incoming applications. The meta-broker interacts with these local schedulers for application execution. Each local scheduler periodically advertise information about available time slots (t_s, t_e, n) to the meta-broker, where t_s and t_e are the start time and end time of the slot respectively and n is the number of CPUs available for the slot.

To facilitate energy-efficient computing, each local scheduler also supplies information about the carbon emission rate, Coefficient of Performance (COP), electricity price, CPU power-frequency relationship, Million Instructions Per Second (MIPS) rating of CPUs at the maximum frequency, and CPU operating frequency range of the data center. The MIPS rating is used to indicate the overall performance of a CPU. All CPUs within a data center are homogeneous, but CPUs can be heterogeneous across data centers. The carbon emission rates are calculated based on the fuel type used in electric power generation. These are published regularly by various government agencies such as U.S. Energy Information Administration (EIA). COP of the data center's cooling system is defined as the amount of cooling delivered per unit of electrical power consumed. COP can be measured by monitoring the energy consumption by various components of the cooling system [128]. Various parameters of CPUs at the data center can be derived experimentally [70].

6.2.1 Data Center Energy Model

The major contributors for the total energy usage in a data center are IT equipments (which consists of servers, storage devices, and network equipment) and cooling systems [159]. Other systems such as lighting are not considered due to their negligible contribution to the total energy usage.

Within a data center, the total energy usage of a server depends on its CPUs, memory, disks, fans, and other components [56]. It is pointed out by Fan et al [56], the energy usage of the server varies depending on the type of workload executed. Since we only consider compute-intensive applications and the CPUs use the largest proportion of energy in a server, it is sufficient in our case to only model CPU energy usage. Thus, for simplicity, we only compute the energy usage of a server based on its CPUs.

The power consumption of a CPU can be reduced by lowering its supply voltage using Dynamic Voltage Scaling (DVS). DVS is an efficient way to manage dynamic power dissipation during computation. The power consumption model of CPUs which are gen-

erally composed of CMOS circuits is given by: $P = \alpha V^2 f + I_{leak} V + P_{short}$, where P is the power dissipation, V is the supply voltage, f is the clock frequency, I_{leak} is the leakage current, and P_{short} is the short circuit power dissipated during the voltage switching process [18][130]. The first term constitutes the dynamic power of the CPU and the second term constitutes the static power. P_{short} is generally negligible in comparison to other terms.

Since the voltage can be expressed as a linear function of frequency in the CMOS logic, the power consumption P_i of a CPU in a data center i is approximated by the following function (similar to previous works [167][33]): $P_i = \beta_i + \alpha_i f^3$, where β_i is the static power consumed by the CPU, α_i is the proportionality constant, and f is the frequency at which the CPU is operating. We use this cubic relationship between the operating frequency and power consumption since this work focuses on compute-intensive workload and to the best of our knowledge, the cubic relationship is the most commonly used metric for CPU power in previous works [167][33]. We also consider that a CPU of data center i can adjust its frequency from a minimum of f_i^{min} to a maximum of f_i^{max} discretely. The frequency levels supported by a CPU typically varies for different CPU architecture. For instance, Intel Pentium M 1.6GHz CPU supports 6 voltages from 0.956V to 1.484V.

The energy cost of the cooling system depends on its COP [114] [157]. COP is an indication for the efficiency of the cooling system, which is defined as the ratio of the amount of energy consumed by CPUs to the energy consumed by the cooling system. However, COP is not constant and varies with the cooling air temperature. We assume that COP will remain constant during a scheduling cycle and data centers will update the meta-broker whenever COP changes. Thus, the total energy consumed by the cooling system in a data center i is given by:

$$E_i^h = \frac{E_i^c}{COP_i} \tag{6.1}$$

where E_i^c is the total energy consumed by CPUs and E_i^h is the total energy consumed by cooling devices. The total energy consumed by data center i can then be approximated by:

$$E_i^{total} = E_i^c + E_i^h = (1 + \tfrac{1}{COP_i})E_i^c = (\tfrac{COP_i+1}{COP_i})E_i^c$$

Therefore, the data center efficiency (DCiE) [164] is given as:

$$DCiE = \tfrac{COP}{COP+1}$$

6.2.2 Relation between Execution Time and CPU Frequency

Since DVS is adopted to scale up/down the CPU frequency, the execution time of an application can significantly vary according to the CPU frequency. However, the decrease in the execution time due to the increase in CPU frequency depends on whether the application is CPU bound or not. For example, if the performance of an application is completely dependent on the CPU frequency, then its execution time will be inversely proportional to the change in the CPU frequency. Thus, the execution time of an application is modelled according to the definition proposed by Hsu et al. [84]:

$$T(f) = T(f^{max}) \times (\gamma^{cpu}(\frac{f^{max}}{f} - 1) + 1) \tag{6.2}$$

where $T(f)$ is the execution time of the application at CPU frequency f, $T(f^{max})$ is the execution time of the application at the maximum CPU frequency f^{max}, and γ^{cpu} is the CPU boundness of the application.

If the value of γ^{cpu} decreases, the CPU boundness of the application will also decrease, which results in potentially more energy reduction by using DVS in servers within a data center (see Section 6.3.2). It is however important to note that the CPU boundness of an application varies based on the CPU architecture, as well as memory and disks. Like many prior studies [53][66], we still use this factor to model the CPU usage intensity of an application in a simple generic manner so as to optimise its energy consumption accordingly. However, for all our experiments, we have used the worst case value of $\gamma^{cpu} = 1$ to analyse the performance of our heuristics.

6.2.3 Problem Description

Table 6.2: Parameters of a Data Center i

Parameter	Notation
Carbon emission rate (kg/kWh)	$r_i^{CO_2}$
Average COP	COP_i
Electricity price ($/kWh)	p_i^e
Data transfer price ($/GB) for upload/download	p_i^{DT}
CPU power	$P_i = \beta_i + \alpha_i f^3$
CPU frequency range	$[f_i^{min}, f_i^{max}]$
Time slots (start time, end time, number of CPUs)	(t_s, t_e, n)

Let a resource provider have N data centers distributed in different locations. All the parameters associated with data center i are given in Table 6.2. Data center i incurs carbon emission based on its carbon emission rate $r_i^{CO_2}$ (kg/kWh). To execute an application, the resource provider has to pay data center i the energy cost and data transfer cost depending on its electricity price p_i^e (\$/kWh) and data transfer price p_i^{DT} (\$/GB) for upload/download respectively. The resource provider then charges fixed prices to the user for executing his application based on the CPU execution price p^c (\$/CPU/hour) and data transfer price p^{DTU} (\$/GB) for the processing time and upload/download respectively.

Let J be the total number of user applications. A user submits her requirements for application j in the form of a tuple $(d_j, n_j, e_{j1}, \ldots, e_{jN}, \gamma_j^{cpu}, (DT)_j)$, where d_j is the deadline to complete application j, n_j is the number of CPUs required for application execution, e_{ji} is the application execution time on the data center i when operating at the maximum CPU frequency, γ_j^{cpu} is the CPU boundness of the application, and $(DT)_j$ is the size of data to be transferred. For simplicity, we assume that users are able to specify their processing requirements (n_j, e_{ji}, and γ_j^{cpu}).

In addition, let f_{ij} be the initial frequency at which CPUs of a data center i operate while executing application j. Hence executing application j on data center i results in the following:

(i) Energy consumption of CPUs

$$E_{ij}^c = (\beta_i + \alpha_i (f_{ij})^3) \times n_j e_{ji} \times (\gamma_j^{cpu} (\frac{f_i^{max}}{f_{ij}} - 1) + 1) \tag{6.3}$$

(ii) Total energy which consist of the cooling system and CPUs

$$E_{ij} = \frac{COP_i + 1}{COP_i} \times E_{ij}^c \tag{6.4}$$

(iii) Energy cost

$$C_{ij}^e = p_i^e \times E_{ij} \tag{6.5}$$

(iv) Carbon emission

$$(CO_2\ E)_{ij} = r_i^{CO_2} \times E_{ij} \tag{6.6}$$

(v) Execution Profit

$$(ProfExec)_{ij} = n_j e_{ji} p^c - C_{ij}^e \tag{6.7}$$

(vi) Data Transfer Profit

$$(ProfData)_{ij} = (DT)_j \times (p^{DTU} - p_i^{DT}) \tag{6.8}$$

(vii) Profit

$$(Prof)_{ij} = (ProfExec)_{ij} + (ProfData)_{ij} \tag{6.9}$$

The carbon emission $(CO_2\ E)_{ij}$ (Equation (6.6)) incurred by application j is computed using the carbon emission rate $r_i^{CO_2}$ of data center i. However, this means that $(CO_2\ E)_{ij}$ only reflects the average carbon emission incurred since $r_i^{CO_2}$ is an average rate. We can only use $r_i^{CO_2}$, since the exact amount of carbon emission produced depends on the type of fuel used to generate the electricity, and no detailed data is available in this regard.

The profit $(Prof)_{ij}$ (Equation (6.9)) gained by the resource provider from the execution of application j on data center i includes the execution profit $(ProfExec)_{ij}$ and input/output data transfer profit $(ProfData)_{ij}$. Studies [62][10] have shown that the ongoing operational costs (such as energy cost) of data centers greatly surpass their one-time capital costs (such as hardware and support infrastructure costs). Hence, when computing the execution profit $(ProfExec)_{ij}$, we assume that the CPU execution price p^c charged by the resource provider to the user already includes the one-time capital costs of data centers, so that we only subtract the ongoing energy cost C_{ij}^e of executing applications from the revenue. The data transfer profit $(ProfData)_{ij}$ is the difference between the cost paid by the user to the provider and the cost incurred for transferring the data to the data center.

The meta-scheduling problem can then be formulated as:

$$\text{Minimise Carbon Emission} \quad = \quad \sum_{i}^{N}\sum_{j}^{J} x_{ij}(CO_2\ E)_{ij} \tag{6.10}$$

$$\text{Maximise Profit} \quad = \quad \sum_{i}^{N}\sum_{j}^{J} x_{ij}(Prof)_{ij} \tag{6.11}$$

Subject to:

(a) Response time of application $j < d_j$

(b) $f_i^{min} < f_{ij} < f_i^{max}$

(c) $\sum_i^N x_{ij} \leq 1$

(d)

$$x_{ij} = \begin{cases} 1 & \text{if application } j \text{ allocated to data center } i \\ 0 & \text{otherwise} \end{cases}$$

The dual objective functions (6.10) and (6.11) of the meta-scheduling problem are to minimise the carbon emission and maximise the profit of a resource provider. Constraint (a) ensures that the deadline requirement of an application is met. But it is difficult to calculate the exact response time of an application since applications have different sizes, require multiple CPUs, and have very dynamic arrival rates [33]. Moreover, this problem maps to the 2-dimensional bin-packing problem which is NP-hard in nature [112]. Hence, we propose various scheduling policies to heuristically approximate the optimum.

6.3 Meta-Scheduling Policies

The meta-broker periodically assigns applications to data centers at a fixed time interval. This enables the meta-broker to potentially make a better selection choice of applications when mapping from a larger pool of applications to the data centers, as compared to during each submission of an application. In each scheduling cycle, the meta-broker collects the information from both data centers and users.

In general, a meta-scheduling policy consists of two phases: 1) mapping phase, in which the meta-broker first maps an application to a data center; and 2) scheduling phase, in which the scheduling of applications is done within the data center, where the required time slots is chosen to execute the application. Depending on the objective of resource provider whether to minimise carbon emission or maximise profit, we have designed various mapping policies which are discussed in the subsequent section. To further reduce the energy consumption within the data center, we have designed a DVS based scheduling policy for the local scheduler of a data center.

6.3.1 Mapping Phase (Across Many Data Centers)

We have designed the following meta-scheduling policies to map applications to data centers depending on the objective of the resource provider:

Minimising Carbon Emission

The following policies optimise the global carbon emission of all data centers while keeping the number of deadline misses low.

- **Greedy Minimum Carbon Emission (GMCE):** Since the aim is to minimise the carbon emission across all the data centers, we want the most number of applications to be executed on data centers with the least carbon emission. Hence applications are sorted by their deadline (earliest first) to reduce the deadline misses, while

data centers are sorted by their carbon emission (lowest first), which is computed as: $r_i^{CO_2} \times \frac{COP_i+1}{COP_i} \times (\beta_i + \alpha_i(f_i^{max})^3)$. Each application is then mapped to a data center in this ordering.

- **Minimum Carbon Emission - Minimum Carbon Emission (MCE-MCE):** MCE-MCE is based on the Min-Min heuristic [110] which has performed very well in previous studies of different environments [15]. The meta-broker first finds the "best" data center for all applications that are considered. Then among these application-data center pairs, the meta-broker selects the "best" pair to map first. Since the aim is to minimise the carbon emission, the "best" pair has the minimum carbon emission $(CO_2\ E)_{ij}$, i.e. minimum fitness value of executing application j on data center i. MCE-MCE has the following steps:

  Step 1: For each application in the list of applications to be mapped, find the data center of which the carbon emission is the minimum, i.e. minimum $(CO_2\ E)_{ij}$ (the first MCE), among all data centers which can complete the application by its deadline. If there is no data center where the application can be completed by its deadline, the application is removed from the list of applications to be mapped.

  Step 2: Among all the application-data center pairs found in Step 1, find the pair that results in the minimum carbon emission, i.e. minimum $(CO_2\ E)_{ij}$ (the second MCE). Then, map the application to the data center, and remove it from the list of applications to be mapped.

  Step 3: Update the available time slots from data centers.

  Step 4: Do Step 1 to 3 again until all applications are mapped.

Maximising Profit

The following policies optimise the global profit of all data centers while keeping the number of deadline misses low.

- **Greedy Maximum Profit (GMP):** Since the aim is to maximise the profit across all the data centers, we want the most number of applications to be executed on data centers with the least energy cost. Hence applications are sorted by their deadline (earliest first) to reduce the deadline misses, while data centers are sorted by their energy cost (lowest first), which is computed as: $p_i^e \times \frac{COP_i+1}{COP_i} \times (\beta_i + \alpha_i(f_i^{max})^3)$. Each application is then mapped to a data center in this ordering.

- **Maximum Profit - Maximum Profit(MP-MP):** MP-MP works in the same way as MCE-MCE. However, since the aim is to maximise the profit, the "best" pair has

the maximum profit $(Prof)_{ij}$, i.e. maximum fitness value of executing application j on data center i. Hence the steps of MP-MP are the same as MCE-MCE, except the following differences:

Step 1: For each application in the list of applications to be mapped, find the data center of which the profit is the maximum, i.e. maximum $(Prof)_{ij}$ (the first MP), among all data centers which can complete the application by its deadline.

Step 2: Among all the application-data center pairs found in Step 1, find the pair that results in the maximum profit, i.e. maximum $(Prof)_{ij}$ (the second MP).

Minimising Carbon Emission and Maximising Profit (MCE-MP)

MCE-MP works in the same way as MCE-MCE. But, since the aim is to minimise the total carbon emission while maximising the total profit across all the data centers, MCE-MP handles the trade-off between carbon emission and profit which may be conflicting. Hence the steps of MCE-MP are the same as MCE-MCE, except the following differences:

Step 1: For each application in the list of applications to be mapped, find the data center of which the carbon emission is the minimum, i.e. minimum $(CO_2\ E)_{ij}$ (the first MCE), among all data centers which can complete the application by its deadline.

Step 2: Among all the application-data center pairs found in Step 1, find the pair that results in the maximum profit, i.e. maximum $(Prof)_{ij}$ (the second MP).

6.3.2 Scheduling Phase (Within a Data Center)

The energy consumption and carbon emission are further reduced within a data center by using DVS at the CPU level to save energy by scaling down the CPU frequency. Thus, before the meta-broker assigns an application to a data center, it decides the time slot in which the application should be executed and the frequency at which the CPU should operate to save energy. But, since a lower CPU frequency can increase the number of applications rejected due to the deadline misses, the scheduling of applications within the data center can be of two types: 1) CPUs run at the maximum frequency (i.e. without DVS) or 2) CPUs run at various frequencies using DVS (i.e. with DVS). It is important to adjust DVS appropriately in order to reduce the number of deadline misses and energy consumption simultaneously.

The meta-broker will first try to operate the CPU at a frequency in the range $[f_i^{min}, f_i^{max}]$ nearest to the optimal CPU frequency $f_i^{opt} = \sqrt[3]{\frac{\beta_i}{2\alpha_i}}$ for $\gamma^{cpu} = 1$. Since the CPU frequency of data center i can only operate in the interval $[f_i^{min}, f_i^{max}]$, we define $f_i^{opt} = f_i^{min}$ if $f_i^{opt} < f_i^{min}$, and $f_i^{opt} = f_i^{max}$ if $f_i^{opt} > f_i^{max}$.

If the deadline of an application will be violated, the meta-broker will scale up the CPU frequency to the next level and then try again to find the free time slots to execute the application. If the meta-broker fails to schedule the application on the data center as no free time slot is available, then the application is forwarded to the next data center for scheduling (the ordering of data centers depends on various policies as described in Section 6.3.1).

6.3.3 Lower Bound and Upper Bound

Due to the NP hardness of the meta-scheduling problem described in Section 6.2.3, it is difficult to find the optimal profit and carbon emission in polynomial time. Thus, to estimate the performance of our scheduling algorithms, we present a lower bound for the carbon emission and an upper bound for the profit of the resource provider respectively. Both bounds are derived based on the principle that we can get the minimum carbon emission or the maximum profit when most of the applications are executed on the most "efficient" data center and also at the optimal CPU frequency. For carbon emission minimisation, the most "efficient" data center incurs the minimum carbon emission for executing applications, while for profit maximisation, the most "efficient" data center results in the minimum energy cost.

For the sole purpose of deriving the lower and upper bounds, we relax three constraints of our system model so as to map the maximal number of applications to the most "efficient" data center. First, we relax the constraint that when an application is executed at the maximum CPU frequency, it will result in the maximum energy consumption. Instead, we assume that even though all applications are executed at the maximum CPU frequency, the actual energy consumed by them for calculating their carbon emission still remains at the optimal CPU frequency using DVS. Second, although the applications considered in the system model are parallel applications with fixed CPU requirements, we relax this constraint to applications that are moldable in the required number of CPUs. Thus, the runtime of applications will decrease linearly when it is scheduled on a larger number of CPUs. This in turn increases the number of applications that can be allocated to the most "efficient" data center with the minimum energy possible. Third, the applications in the system model are arriving dynamically in many different scheduling cycles, but for deriving the bounds, all applications are considered in only one scheduling cycle and mapped to data centers. This forms the best ideal bound scenario, where all the incoming applications are known in advance. Hence the actual dynamic scenario definitely has worse performance than that of the ideal bound scenario.

It is important to note that the bounds of carbon emission and profit obtained with these three assumptions are unreachable loose bounds of the system model. This is because data centers will be executing the maximum possible workload with 100% utilisation of their

CPUs, while the least possible energy consumption is still considered for the purpose of comparison.

Let TWL be the total workload scheduled, TCE be the total carbon emission, and TP be the total profit. The lower bound for the carbon emission is derived through the following steps:

Step 1: Applications are sorted by their deadline (earliest first) to reduce the deadline misses, while data centers are sorted by their carbon emission (lowest first), which is computed as: $r_i^{CO_2} \times \frac{COP_i+1}{COP_i} \times (\beta_i + \alpha_i (f_i^{max})^3)$. Each application is then mapped to a data center in this ordering.

Step 2: For each application j, search for a data center i, starting from the most "efficient" one, where the application j can be scheduled without missing its deadline when running at the maximum CPU frequency.

Step 3: If a data center i is not found, then application j will be removed from the list of potential applications. Go to step 2 to schedule other applications.

Step 4: If a data center i is found, application j is assigned to it and molded such that there is no fragmentation in the schedule of data center i for executing applications.

Step 5: TWL += $n_j \times e_{ji}$

Step 6: TCE += $r_i^{CO_2} \times \frac{COP_i+1}{COP_i} \times$ (Power consumption of the CPU at optimal CPU frequency) $\times n_j \times$ (Execution time of application j at optimal CPU frequency)

Step 7: TP += $(1 - (p_i^e \times \frac{COP_i+1}{COP_i} \times$ (Power consumption of the CPU at optimal CPU frequency))) $\times n_j \times$ (Execution time of application j at optimal CPU frequency)

Step 8: Repeat from Step 2 until all applications are scheduled.

$\frac{TCE}{TWL}$ will be the lower bound of the average carbon emission due to the execution of all applications across multiple data centers of the resource provider.

To derive the upper bound for the profit, the steps remain the same, except the following differences:

- In Step 1, data centers are sorted by their energy cost (lowest first), which is computed as: $p_i^e \times \frac{COP_i+1}{COP_i} \times (\beta_i + \alpha_i (f_i^{max})^3)$.
- $\frac{TP}{TWL}$ will be the upper bound of the average profit.

6.4 Performance Evaluation

Configuration of Applications: We use workload traces from Feitelson's Parallel Workload Archive (PWA) [57] to model the workload. Since this chapter focuses on studying the requirements of users with compute-intensive applications, the PWA meets our objective by providing workload traces that reflect the characteristics of real parallel applications. Our experiments utilise the first week of the LLNL Thunder trace (January 2007 to June 2007). The LLNL Thunder trace from the Lawrence Livermore National Laboratory (LLNL) in USA is chosen due to its highest resource utilisation of 87.6% among available traces to ideally model a heavy workload scenario. From this trace, we obtain the submit time, requested number of CPUs, and actual runtime of applications. We set the CPU boundness of all workload as 1 (i.e. $\gamma^{cpu} = 1$) to examine the worst case scenario of CPU energy usage. We use a methodology proposed by Irwin et al. [88] to synthetically assign deadlines through two classes namely Low Urgency (LU) and High Urgency (HU).

An application j in the LU class has a high ratio of $deadline_j/runtime_j$ so that its deadline is definitely longer than its required runtime. Conversely, an application j in the HU class has a deadline of low ratio. Values are normally distributed within each of the high and low deadline parameters. The ratio of the deadline parameter's high-value mean and low-value mean is thus known as the high:low ratio. In our experiments, the deadline high:low ratio is 3, while the low-value deadline mean and variance is 4 and 2 respectively. In other words, LU applications have a high-value deadline mean of 12, which is 3 times longer than HU applications with a low-value deadline mean of 4. The arrival sequence of applications from the HU and LU classes is randomly distributed.

Configuration of Data Centers: We model 8 data centers with different configurations as listed in Table 6.3. Carbon emission rates and electricity prices at various data center locations are averages over the entire region and derived from the data published by US Department of Energy [161] and Energy Information Administration [160].

Wang and Lu [167] also focus on the similar problem of considering the energy consumption of heterogeneous CPUs within a data center. Thus, power parameters (i.e. CPU power factors and frequency level) of the CPUs at different data centers are derived based on the experimental data from Wang and Lu's work [167]. The values of α and β are set such that the ratio of static power and dynamic power can cover a wide variety of CPUs.

Current commercial CPUs only support discrete frequency levels, such as the Intel Pentium M 1.6GHz CPU which supports 6 voltage levels. We consider discrete CPU frequencies with 5 levels in the range $[f_i^{min}, f_i^{max}]$. For the lowest frequency f_i^{min}, we use the same value used by Wang and Lu [167], i.e. f_i^{min} is 37.5% of f_i^{max}.

To increase the utilisation of data centers and reduce the fragmentation in the scheduling of parallel applications, the local scheduler at each data center uses Conservative Backfilling with advance reservation support as proposed by Mu'alem and Feitelson [115].

Table 6.3: Characteristics of Data Centers

Location	Carbon Emission Rate[a] (kg/kWh)	Electricity Price[b] ($/kWh)	CPU Power Factors		CPU Frequency Level		Number of CPUs
			β	α	f_i^{max}	f_i^{opt}	
New York, USA	0.389	0.15	65	7.5	1.8	1.630324	2050
Pennsylvania, USA	0.574	0.09	75	5	1.8	1.8	2600
California, USA	0.275	0.13	60	60	2.4	0.793701	650
Ohio, USA	0.817	0.09	75	5.2	2.4	1.93201	540
North Carolina, USA	0.563	0.07	90	4.5	3.0	2.154435	600
Texas, USA	0.664	0.1	105	6.5	3.0	2.00639	350
France	0.083	0.17	90	4.0	3.2	2.240702	200
Australia	0.924	0.11	105	4.4	3.2	2.285084	250

[a]Carbon emission rates are derived from a US Department of Energy (DOE) document (Appendix F-Electricity Emission Factors 2007) [161].
[b]Electricity prices are average commercial prices till 2007 based on a US Energy Information Administration (EIA) report [160].

The meta-broker schedules applications periodically at each scheduling cycle of 50 seconds, which is to ensure that the meta-broker can receive at least one application in every scheduling cycle.

The COP (power usage efficiency) value of data centers is randomly generated using a uniform distribution between $[0.6, 3.5]$ as indicated in the study conducted by Greenberg et al. [73]. To avoid the energy cost of a data center exceeding the revenue generated by the resource provider, the CPU execution price charged by the provider to the user is fixed at \$0.40/CPU/hour which is approximately twice of the maximum energy cost at a data center.

Performance Metrics: We observe the performance from both user and provider perspectives. From the provider perspective, four metrics are necessary to compare the policies: average energy consumption, average carbon emission, profit gained, and workload executed. The *average energy consumption* compares the amount of energy saved by using different scheduling algorithms, whereas the *average carbon emission* compares its corresponding environmental impact. Since minimising the carbon emission can affect a resource provider economically by decreasing its profit, we have considered the *profit gained* as another metric to compare different algorithms. It is important to know the effect of various meta-scheduling policies on energy consumption, since higher energy consumption is likely to generate more carbon emission for worse environmental impact and incur more energy cost for operating data centers.

From the user perspective, we observe the performance of varying: 1) urgency class and 2) arrival rate of applications. For the urgency class, we use various percentages (0%, 20%, 40%, 60%, 80%, and 100%) of HU applications. For instance, if the percentage of HU applications is 20%, then the percentage of LU applications is the remaining 80%. For the arrival rate, we use various factors (10 (low), 100 (medium), 1000 (high), and 10000 (very high)) of submit time from the trace. For example, a factor of 10 means an application with a submit time of 10s from the trace now has a simulated submit time of 1s. Hence, a higher factor represents higher workload by shortening the submit time of applications.

Experimental Scenarios: To comprehensively evaluate the performance of our algorithms, we examine various experimental scenarios that are classified as:

(a) **Evaluation without data transfer cost (Section 6.5.1):** In the first set of experiments (Section 6.5.1), we evaluate the importance of our mapping policies which consider global factors such as carbon emission rate, electricity price and data center efficiency. In these experiments, we also evaluate the effectiveness of exploiting local minima in our local DVS policy (Section 6.5.1 and Section 6.5.1). Then, in the next set of experiments (Section 6.5.1), we compare the performance of our proposed algorithms with the lower bound (carbon emission) and the upper bound

(profit).

To evaluate the overall best among the proposed algorithms, we conduct more experiments by varying different factors which can affect their performance:

- Impact of urgency and arrival rate of applications (Section 6.5.1)
- Impact of carbon emission rate (Section 6.5.1)
- Impact of electricity price (Section 6.5.1)
- Impact of data center efficiency (Section 6.5.1)

(b) **Evaluation with data transfer cost (Section 6.5.2):** In the last set of experiments (Section 6.5.2), we examine how the data transfer cost affect the performance of our algorithms.

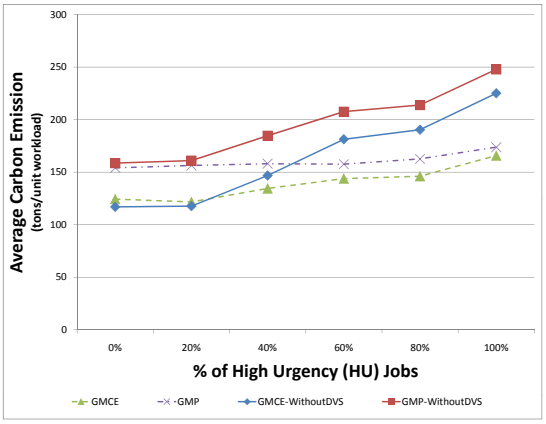
6.5 Analysis of Results

6.5.1 Evaluation without Data Transfer Cost

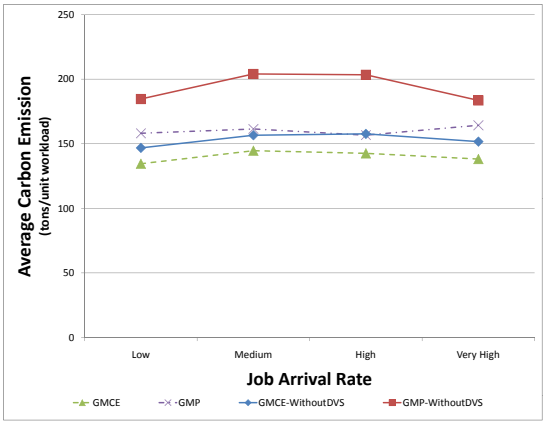
Effect of Mapping Policy and DVS

As discussed in Section 6.3, our meta-scheduling policies are designed to save energy at two phases, first at the mapping phase and then at the scheduling phase. Hence, in this section, we examine the importance of each phase in saving energy. These experiments also answer the question why we require special energy saving schemes at two phases.

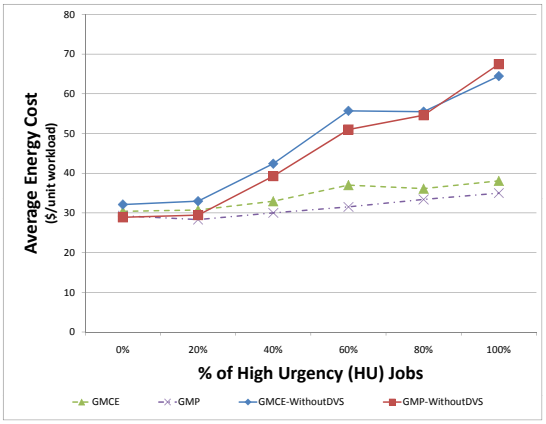
First, we examine the importance of considering the global factors at the mapping phase by comparing meta-scheduling policies without the energy saving feature at the local scheduling phase, i.e. DVS is not available at the local scheduler. Hence, we name the without-DVS version of the carbon emission based policy (GMCE) and profit based policy (GMP) as GMCE-WithoutDVS and GMP-WithoutDVS respectively. The results in Figure 6.2 shows that the consideration of various global factors not only decrease the carbon emission, but also decrease the overall energy consumption. For various urgency of applications (Figure 6.2(a)), GMCE-WithoutDVS can prevent up to 10% carbon emission over GMP-WithoutDVS. For various arrival rate of applications (Figure 6.2(b)), GMCE-WithoutDVS can produce up to 23% less carbon emission than GMP-WithoutDVS. The corresponding difference in energy cost (Figure 6.2(c) and 6.2(d)) between them is very little (about 0–6%). This is because with the decrease in energy consumption due to the execution of HPC workload, both carbon emission and energy cost will automatically decrease. This trend still remains by comparing GMCE and GMP, both of which uses DVS at the scheduling phase.



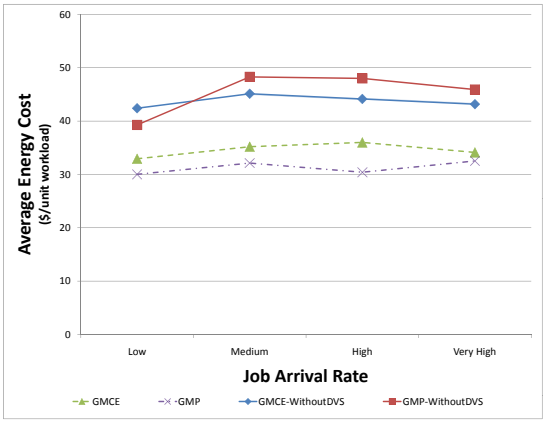
(a) Carbon Emission VS Urgency



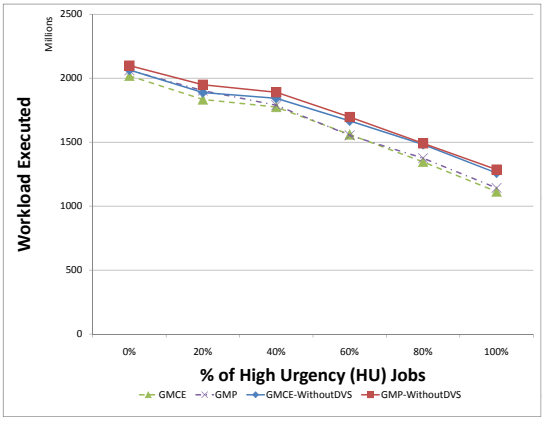
(b) Carbon Emission VS Arrival Rate



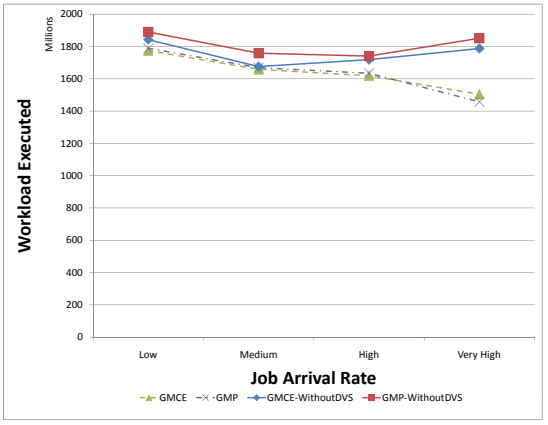
(c) Energy Cost VS Urgency



(d) Energy Cost VS Arrival Rate



(e) Workload Executed VS Urgency



(f) Workload Executed VS Arrival Rate

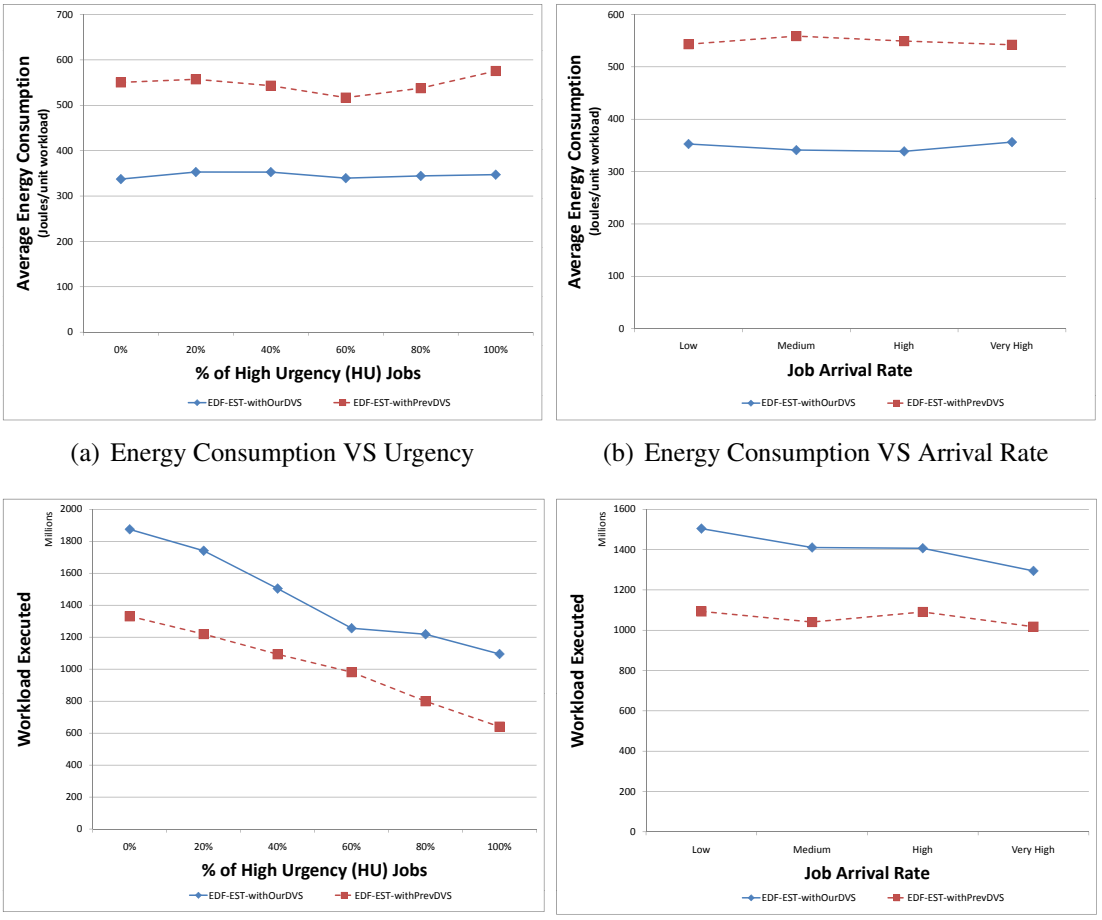
Figure 6.2: Effect of Mapping Policy and DVS

Next, we examine the impact of the scheduling phase on energy consumption by comparing meta-scheduling policies with DVS (GMCE and GMP) and without DVS (GMCE-WithoutDVS and GMP-WithoutDVS). With DVS, the energy cost (Figure 6.2(c)) to execute HPC workload has been reduced on average by 33% when we compare GMP with GMP-withoutDVS. With the increase in HU applications, the gap is increasing and we can get almost 50% decrease in energy cost as shown in Figure 6.2(c). With the increase in arrival rate, we get a consistent 25% gain in energy cost by using DVS (Figure 6.2(d)). The carbon emission is also reduced further on average by 13% with the increase in urgent applications as shown in Figure 6.2(a). With the increase in arrival rate, the HPC workload executed is decreasing in the case of policies using DVS as can be observed from Figure 6.2(f). This is because the execution of applications at lower CPU frequency results in more rejection of urgent applications when the arrival rate is high. Thus, HPC workload executed in the case of policies without DVS is almost the same even when the arrival rate is very high.

Exploiting Local Minima in DVS

We want to highlight the importance of exploiting local minima in the DVS function while scheduling within a data center. But, to correctly highlight the difference in DVS performance for the scheduling phase of the meta-scheduling policy, we need an independent policy (which is not linked to our proposed polices) for the mapping phase. Hence, we use EDF-EST, where the applications are ordered based on Earliest Deadline First (EDF), while the data centers are ordered based on Earliest Start Time (EST). We name our proposed DVS as EDF-EST-withOurDVS that exploits the local minima in the DVS function. Our proposed DVS is compared to a previously proposed DVS named as EDF-EST-withPrevDVS, in which the CPU frequency is scaled up linearly between $[f^{min}, f^{max}]$ [167][97].

Figure 6.3 shows that EDF-EST-withOurDVS has not only outperformed EDF-EST-withPrevDVS by saving about 35% of energy, but also executed about 30% more workload. This is because EDF-EST-withPrevDVS tries to run applications at the minimum CPU frequency f^{min} which may not be the optimal frequency. Thus, it is clear from here that an application executed at f^{min} may not lead to the least energy consumption due to the presence of local minima. Moreover, executing applications at a lower frequency results in a lower acceptance of applications since less CPUs are available. Thus, it is important to exploit such characteristics when designing the scheduling policy within a data center.



(a) Energy Consumption VS Urgency

(b) Energy Consumption VS Arrival Rate

(c) Workload Executed VS Urgency

(d) Workload Executed VS Arrival Rate

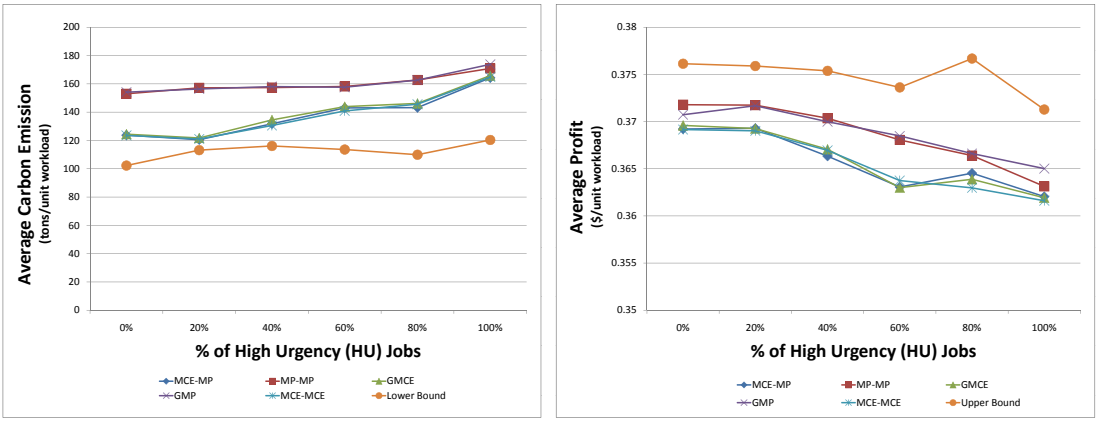
Figure 6.3: Exploiting Local Minima in DVS

Comparison of Lower Bound and Upper Bound

To evaluate the performance of our algorithms in terms of carbon emission reduced and profit gained by the resource provider, we compare our algorithms with the theoretically unreachable bound.

Figure 6.4 shows how different policies closely perform to the lower bound of average carbon emission and the upper bound of average profit. In Figure 6.4(a), the difference in average carbon emission for carbon emission based policies (GMCE, MCE-MCE, and MCE-MP) and the lower bound is less than about 16% which becomes less than about 2% in the case of 20% HU applications. On the other hand, in Figure 6.4(b), the difference in average profit for profit based policies (GMP and MP-MP) and the upper bound is less than about 2% which becomes less than about 1% in the case of 40% of HU applications. Hence, in summary, our carbon emission based and profit based policies perform within about 16% and 2% of the optimal carbon emission and profit respectively.

In Figure 6.4(a) and 6.4(b), with the increase in HU applications, the difference be-



(a) Carbon Emission (b) Profit

Figure 6.4: Comparison of Lower Bound and Upper Bound

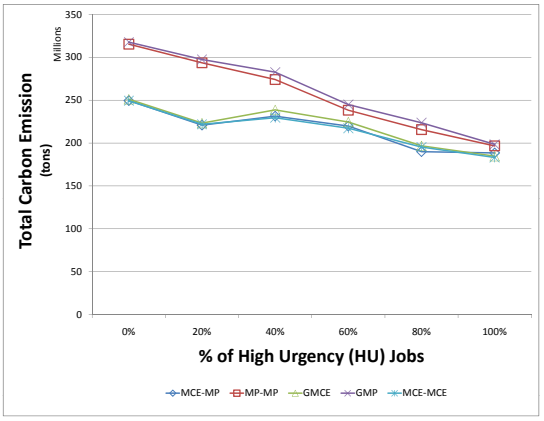
tween the lower/upper bounds and various policies is increasing. This is due to the increase in looseness of the bounds with the increase in HU applications. To avoid deadline misses with a higher number of HU applications, our proposed policies schedule more applications at higher CPU frequency which results in higher energy consumption. This in turn leads to an increase in the carbon emission and decrease in the profit. Whereas, for computing the lower/upper bounds, we only consider energy consumption at the optimal CPU frequency. Thus, the effect of urgency on the bounds is not as considerable as in our policies. This explains why our policies are closer to the bounds for a lower number of HU applications.

Impact of Urgency and Arrival Rate of Applications

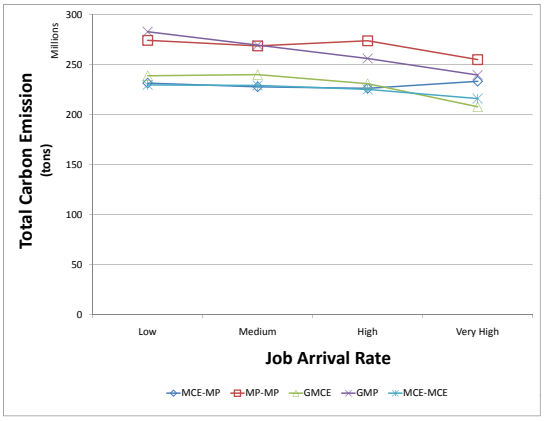
Figure 6.5 shows how the urgency and arrival rate of applications affects the performance of carbon emission based policies (GMCE, MCE-MCE, and MCE-MP) and profit based policies (GMP and MP-MP). The metrics of total carbon emission and total profit are used since the resource provider needs to know the collective loss in carbon emission and gain in profit across all data centers.

When the number of HU applications increases, the total profit of all policies (Figure 6.5(c)) decreases almost linearly by about 45% from 0% to 100% HU applications. Similarly, there is also a drop in total carbon emission (Figure 6.5(a)). This fall in total carbon emission and total profit is due to the lower acceptance of applications as observed in Figure 6.5(e). In Figure 6.5(a), the decrease in total carbon emission for profit based policies (GMP and MP-MP) is much more than that of carbon emission based policies (MCE-MP, GMCE, and MCE-MCE). This is because carbon emission based policies schedule applications on more carbon-efficient data centers.

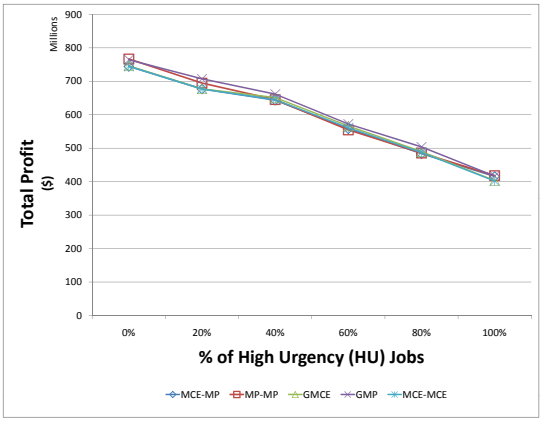
Likewise, the increase in arrival rate also affects the total carbon emission (Figure 6.5(b))



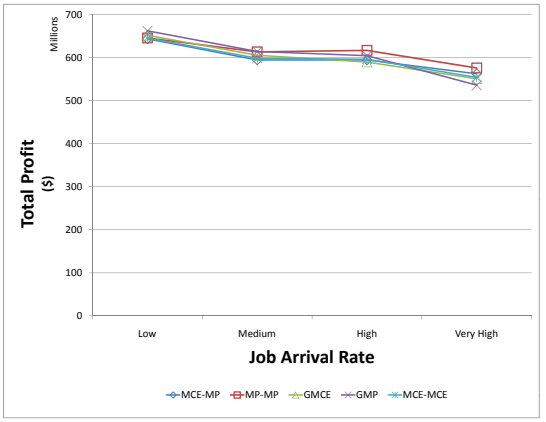
(a) Carbon Emission VS Urgency



(b) Carbon Emission VS Arrival Rate

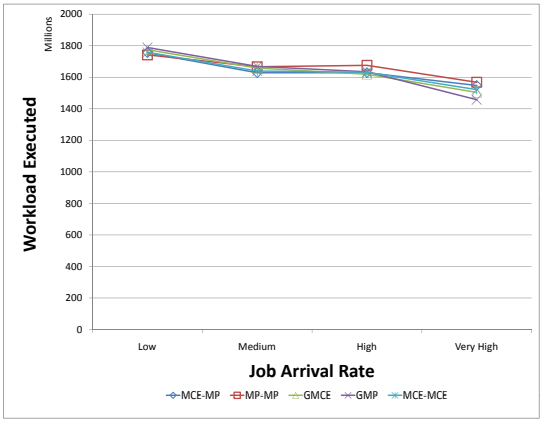


(c) Profit VS Urgency



(d) Profit VS Arrival Rate

(e) Workload Executed VS Urgency



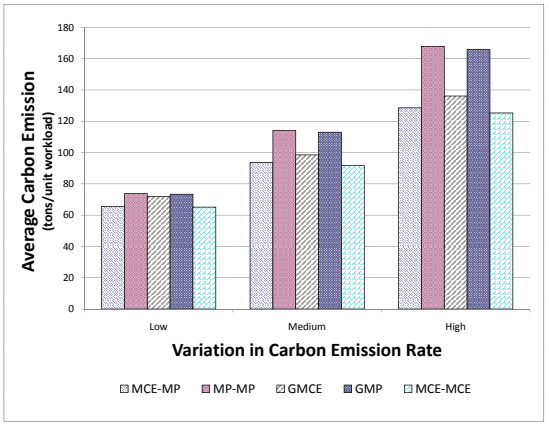
(f) Workload Executed VS Arrival Rate

Figure 6.5: Impact of Urgency and Arrival Rate of Applications

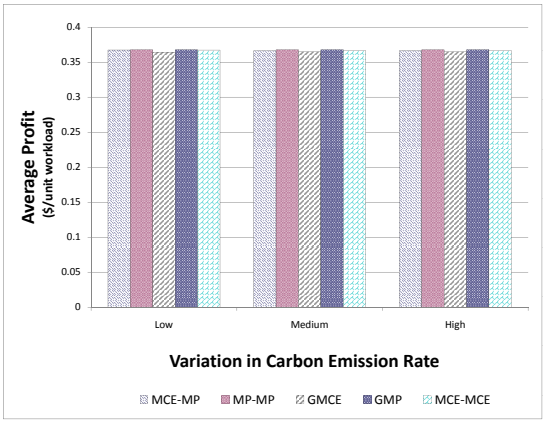
and total profit (Figure 6.5(d)). As more applications are submitted, fewer applications can be accepted (Figure 6.5(f)) since it is harder to satisfy their deadline requirement when workload is high.

Impact of Carbon Emission Rate

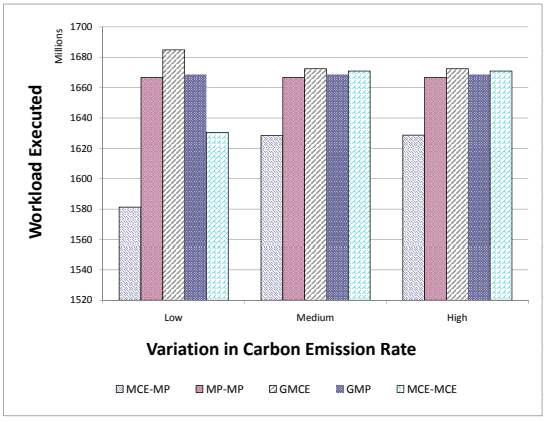
To examine the impact of carbon emission rate in different locations on our policies, we vary the carbon emission rate, while keeping all other factors such as electricity price as the same. Using normal distribution with $mean = 0.2$, random values are generated for the following three classes of carbon emission rate across all data centers as: A) Low variation (low) with $standard\ deviation = 0.05$, B) Medium variation (medium) with $standard\ deviation = 0.2$, and C) High variation (high) with $standard\ deviation = 0.4$. All experiments are conducted at medium job arrival rate with 40% of HU applications.



(a) Carbon Emission



(b) Profit



(c) Workload Executed

Figure 6.6: Impact of Carbon Emission Rate

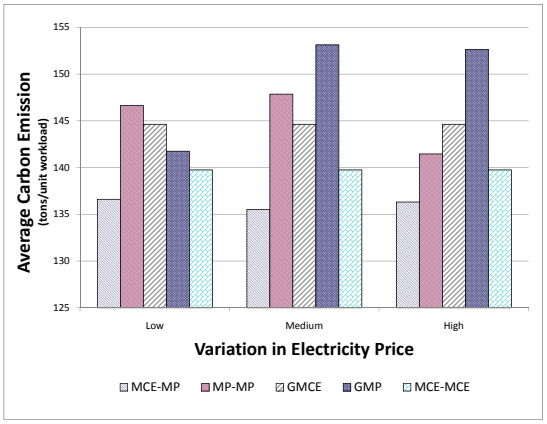
The performance of all policies is similar for all three cases of carbon emission rate. For example, in Figure 6.6(a), the carbon emission of profit based policies (GMP and

MP-MP) is always higher than carbon emission based policies (GMCE, MCE-MCE, and MCE-MP). Similarly, for profit (Figure 6.6(b)), all profit based policies perform better than all carbon emission based policies. For instance, in Figure 6.6(a), the difference in carbon emission of MCE-MCE and MP-MP is about 12% for low variation, which increases to 33% for high variation. On the other hand, in Figure 6.6(b), the corresponding decrease in profit is almost negligible and is less than 1% for both the low and high variation case. Moreover, by comparing MCE-MCE and MP-MP in Figure 6.6(c), the amount of workload executed by MCE-MCE is slightly higher than MP-MP. Thus, for the case of high variation in carbon emission rate, resource providers can use carbon emission based policies such as MCE-MCE to considerably reduce carbon emission with almost negligible impact on their profit. For minimising carbon emission, MCE-MCE is preferred over GMCE since the latter leads to lower profit due to the scheduling of more applications on data centers with higher electricity price.

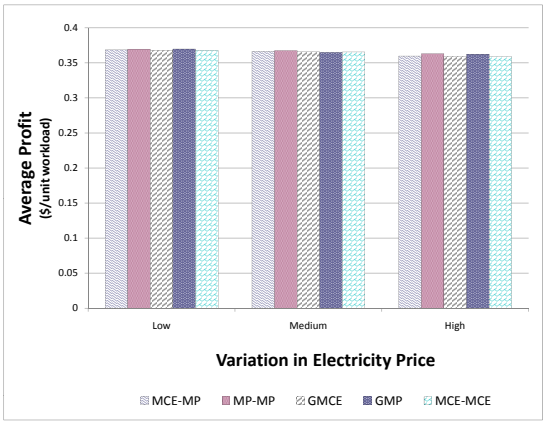
Impact of Electricity Price

To investigate the impact of electricity price in different locations on our policies, we vary the electricity price, while keeping all other factors such as carbon emission rate as the same. Using normal distribution with $mean = 0.1$, random values are generated for the following three classes of electricity price across all data centers as: A) Low variation (low) with $standard\ deviation = 0.01$, B) Medium variation (medium) with $standard\ deviation = 0.02$, and C) High variation (high) with $standard\ deviation = 0.05$. All experiments are conducted at medium job arrival rate with 40% of HU applications.

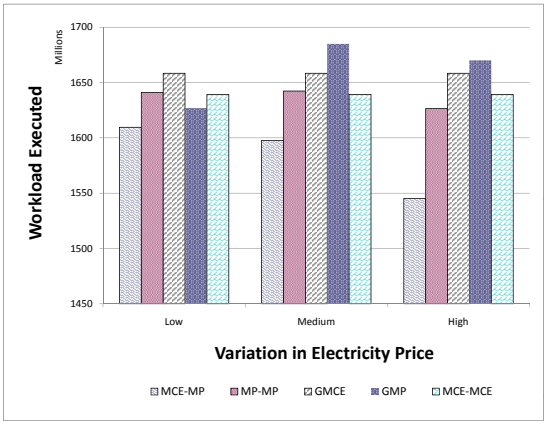
The variation in electricity price affects the performance of profit based policies (GMP and MP-MP) in terms of carbon emission (Figure 6.7(a)) and workload executed (Figure 6.7(c)), while carbon emission based policies (GMCE, MCE-MCE and MCE-MP) are not affected. But, the profit of all policies decrease more as the variation of electricity price increases (Figure 6.7(b)) due to the subtraction of energy cost from profit. For high variation in electricity price, there is not much difference (about 1.4%) in carbon emission between MP-MP and MCE-MCE (Figure 6.7(a)). Hence, resource providers can use MP-MP which gives slightly better average profit than carbon emission based policies (GMCE, MCE-MCE and MCE-MP). On the other hand, for cases when the variation in electricity price is not high, providers can use carbon emission based policies such as MCE-MCE and MCE-MP to reduce about 5-7% of carbon emission by sacrificing less than 0.5% of profit.



(a) Carbon Emission



(b) Profit



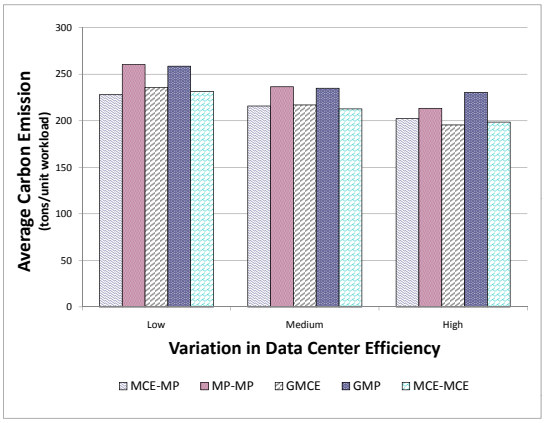
(c) Workload Executed

Figure 6.7: Impact of Electricity Price

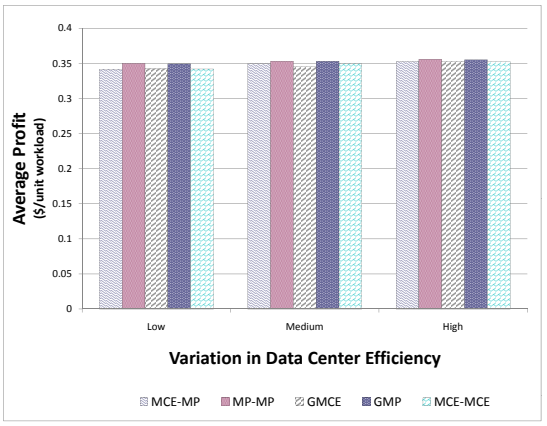
Impact of Data Center Efficiency

To study the impact of data center efficiency in different locations on our policies, we vary the data center efficiency $= \frac{COP}{COP+1}$, while keeping all other factors such as carbon emission rate as the same. Using normal distribution with $mean = 0.4$, random values are generated for the following three classes of data center efficiency across all data centers as: A) Low variation (low) with $standard\ deviation = 0.05$, B) Medium variation (medium) with $standard\ deviation = 0.12$, and C) High variation (high) with $standard\ deviation = 0.2$. All experiments are conducted at medium job arrival rate with 40% of HU applications.

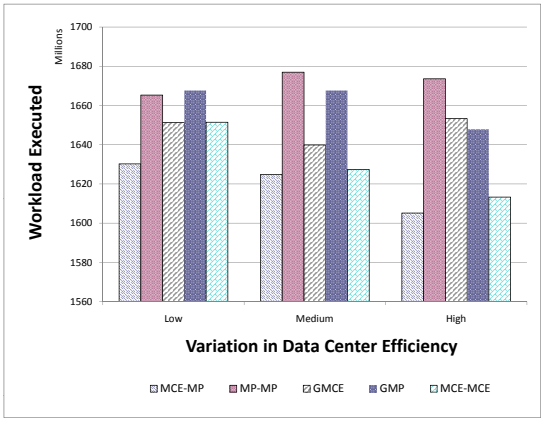
Figure 6.8(a) shows carbon emission based policies (GMCE, MCE-MCE and MCE-MP) achieve the lowest carbon emission with almost equal values. MCE-MCE performs better than MCE-MP by scheduling more HPC workload (Figure 6.8(c)) while achieving similar profit (Figure 6.8(b)). But when the variation in data center efficiency is high, GMCE can execute much higher workload (Figure 6.8(c)) than MCE-MCE and MCE-



(a) Carbon Emission



(b) Profit



(c) Workload Executed

Figure 6.8: Impact of Data Center Efficiency

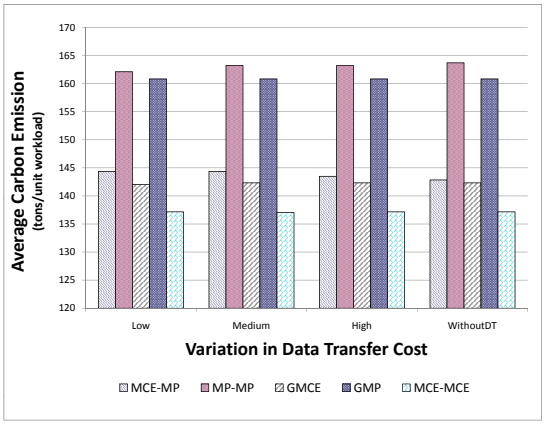
MP while achieving only slightly less profit than profit based policies (GMP and MP-MP) (Figure 6.8(b)). Thus, resource providers can use GMCE to decrease the carbon emissions across their data centers without significant profit loss.

6.5.2 Evaluation with Data Transfer Cost

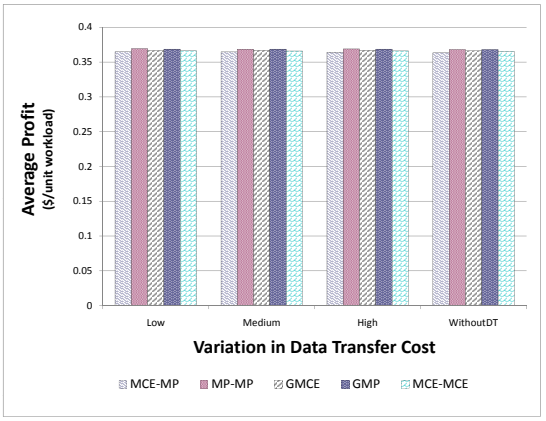
Impact of Data Transfer Cost

The data transfer cost of the resource provider varies across different data centers. Thus, to study the impact of data transfer cost on our policies, we vary the data transfer cost while keeping all other factors such as carbon emission rate and electricity price as the same. For this set of experiments, the resource provider charges the user a fixed price of $0.17/GB for data transfer up to 10TB, which is derived from Amazon EC2 [6]. Since this thesis focuses on compute-intensive parallel applications, the maximum data transfer size of an application is limited to 10TB. The data transfer size of an application is varied between

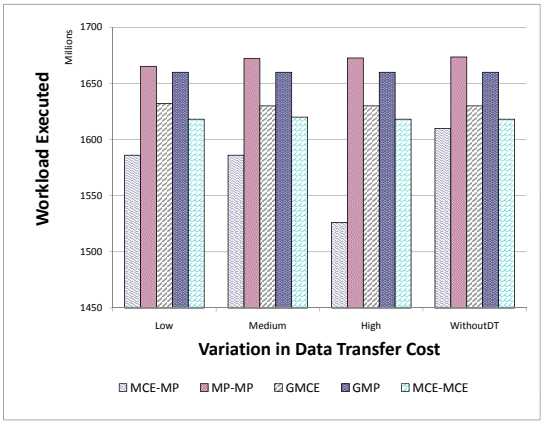
$[0, 10]$TB using uniform distribution. The data transfer cost that the resource provider has to incur is varied between $\$[0, 0.17]$ using normal distribution with $mean = 0.4 * 0.17$. Random values are generated for the following three classes of data transfer cost across all data centers as: A) Low variation (low) with $standard\ deviation = 0.05$, B) Medium variation (medium) with $standard\ deviation = 0.12$, and C) High variation (high) with $standard\ deviation = 0.2$. All experiments are conducted at medium job arrival rate with 20% of HU applications.



(a) Carbon Emission



(b) Profit



(c) Workload Executed

Figure 6.9: Impact of Data Transfer Cost

Figure 6.9 shows how the average carbon emission and profit will be affected due to data transfer cost in comparison to the case when data transfer cost is not considered (as indicated by WithoutDT). The relative performance of all policies has remained almost the same even with data transfer cost. For instance, in Figure 6.9(a) and 6.9(c), MP-MP results in the maximum average carbon emission, while MCE-MCE results in the minimum carbon emission. This is because of the compute-intensive workload, whereby the impact of data transfer cost is negligible in comparison to the execution cost. There is only a slight increase in the average profit (Figure 6.9(b)) due to the additional profit

gained by the resource provider from the transfer of data.

6.6 Summary

The high usage of energy has become a major concern due to its large share in maintenance cost of data centers. Especially, resource providers need a high amount of electricity to run and maintain their computational resources, in order to guarantee the best service level for the customer. Although this importance has been emphasised in a lot of research literature, the combined approach of analysing the profit and energy sustainability in the resource allocation process has not been taken into consideration.

Table 6.4: Summary of Heuristics with Comparison Results

Meta-Scheduling Policy	Description	Time Complexity	Overall Performance				
			HU Jobs	Arrival Rate	Carbon Emission Rate	Data Center Efficiency	Energy Cost
GMCE	Greedy (Carbon Emission)	$O(NJ)$	Bad	Bad	Bad	Best (high)	Bad
MCE-MCE	Two-phase Greedy (Carbon Emission)	$O(NJ^2)$	Good (low)	Good (low)	Best (high)	Okay (low)	Good (low)
GMP	Greedy (Profit)	$O(NJ)$	Okay (high)	Okay (high)	Bad (low)	Bad (high)	Bad
MP-MP	Two-phase Greedy (Profit)	$O(NJ^2)$	Good (high)	Bad (Carbon Emission), Best (Profit)	Good (low)	Best (low)	Good (high)
MCE-MP	Two-phase Greedy (Carbon Emission and Profit)	$O(NJ^2)$	Best (low)	Good (high)	Okay	Okay	Best (low)

We outline how the meta-broker manages the resource allocation across multiple locations to maximise the provider's utility (profit). The overall meta-scheduling problem is formulated as an optimisation problem with dual objective functions. Due to its NP hard nature, several heuristic policies are proposed, and compared for profit maximisation in various test cases. In some cases, the policies perform very well with only almost 1% away from the upper bound of profit. By introducing DVS and hence lowering the supply voltage of CPUs, the energy cost for executing HPC workloads can be reduced by 33% on average. Applications run on CPUs with a lower frequency than expected, but they

still meet the required deadlines. The limitation of carbon emission can be forced by governments to comply with certain threshold values [96]. In such cases, resource providers should focus on reducing carbon emission in addition to minimising energy consumption.

We identify that policies, such as MCE-MCE, can help providers to reduce their emission while almost maintaining their profit. If providers face a volatile electricity price, the MP-MP policy will lead to a better outcome. Depending on the environmental and economic constraints, resource providers can selectively choose different policies to efficiently allocate their resources to meet customers' requests. The characteristics and performance of each meta-scheduling policy are summarised in Table 6.4, where "low" and "high" represent the scenario for which the overall performance of the policy is given. For instance, GMCE performs the best when the variation in data center efficiency is high, while MCE-MP performs the best when the variation in energy cost is low, or when there is a low number of HU applications. We observe that the impact of data transfer cost is minimal for the compute-intensive applications that we have considered. However, our model has explicitly considered the data transfer cost, and thus can be used for data-intensive applications as well.

Hence, this chapter addresses the problem of profit maximisation for the resource provider having multiple resource sites (data centers) by intelligent meta-scheduling of user applications with QoS requirements. In the next chapter, we will investigate the market-oriented meta-scheduling problem in the third scenario i.e. to maximise the utility of both Grid users and resource providers.

Chapter 7

Meta-Scheduling to Enhance All Grid Players' Utility

In previous chapters, we developed market-oriented meta-scheduling algorithms for two scenarios where matching of multiple applications and resources is required. In this chapter, we aim to apply the market-oriented meta-scheduling in order to maximise simultaneously both Grid users' and resource providers' utility. To achieve this objective, valuation metrics are designed and presented for both user applications and resources. The valuation metrics commodify the complex resource requirements of the users and the capabilities of available computational resources. An analytical model of the meta-scheduling problem in Grids is presented to ensure correctness of our mechanism, and to estimate effect on other crucial system-based metrics such as slowdown and waiting time. Then, at the end of the chapter, a simulation study discusses the performance of our presented mechanism in comparison to both existing market-based and traditional meta-scheduling algorithms. The results show that our meta-scheduling mechanism not only satisfies more user requirements (users' utility) than others, but also improves system utilisation (providers' utility) through load balancing.

7.1 Motivation

As discussed in Chapter 1 and 2, the previously proposed solutions for meta-scheduling in Grid computing focused more on improving system-centric performance metrics such as utilisation, average load and turnaround time for user applications [136]. They were not designed to cater to the sophisticated QoS needs of an application, particularly when the demand for resources exceeded the supply. Thus, in recent years, a number of researchers have explored application of market-based models to address user requirements in meta-scheduling. Auctions have been particularly preferred [9, 104] as they provide immense

flexibility to participants to specify their valuations for applications and resources. However, these systems have limitations [168]. It is difficult for the users to come up with a valuation corresponding to their utility function. In a Grid with changing availability of resources, it is difficult to determine resource and application valuations accurately. Also, users with low valued and urgent requirements may be starved by those with higher value. Auctions are applied to commodities that are comparable to one another. However, the parallel applications having fixed processor requirements are not comparable, and cannot be commodified easily. Hence, the problem is to design valuation metrics that commoditize resource requirements and availability so as to take advantage of the efficiency of auction mechanisms [146]. Thus, we need new scheduling mechanisms that are not only efficient and ensure better performance of Grid resources, but also take into account user interests, resource valuation and demand, and allocate resources fairly to user applications by reducing the starvation of low valued and urgent applications.

Therefore, in this chapter, we present a novel market-oriented meta-scheduling mechanism which takes inspiration from auction principles to allocate resources in global Grid environment. The objective of our meta-scheduling mechanism is to maximise both the users' utility by reducing starvation of applications, and the resource providers' utility by equally distributing the load, while minimising affect on other crucial metrics such as waiting time and slowdown.

7.2 System Model

We consider a Grid with m resource sites, $R_1, R_2..R_m$ having k job queues. Resource sites supply information about available slots, load and waiting times of each queue to the meta-broker at regular time intervals. A *slot* is a unit of resource allocation which is described by a start time, a finish time, and the number of processors available for that duration. A resource site also supplies an initial valuation for running an application in a queue on that resource. The utility of of the resource provider is considered in the terms of fairness in receiving the workload for execution. Thus, the objective of the meta-broker is to equally distribute the load across all the resource sites.

The objective of the users is to have their applications completed by a deadline. It is assumed that deadlines are hard, i.e. a user will benefit only if his/her application is executed by its deadline. Users will also provide an initial valuation of the application to the meta-broker.

7.3 Double Auction-Inspired Meta-scheduling (DAM)

Auction-based mechanisms have been the subject of many previous studies. Grosu *et al.* [74] compare resource allocation protocols using First-Price, Second-Price, Vickery

and Double Auction (DA). They show that DA favours both users and resources, while the first-price auction is biased towards resources and the Vickery auction favours users. Therefore, we have opted to use principles of DA (*aka* Call auction) for designing our meta-scheduling mechanism.

In a typical Call auction, sellers and buyers submit offers (*asks*) and requests (*bids*) respectively to an auctioneer who continually ranks them from highest to lowest in order to generate demand and supply profiles. From the profiles, the maximum quantity exchanged can be determined by matching asks, starting with the lowest price and moving up, with the bids, starting with the highest price and moving down. This format allows buyers to make offers, and sellers to accept those offers at any particular moment even though the matching of asks and bids is done after specific scheduling intervals.

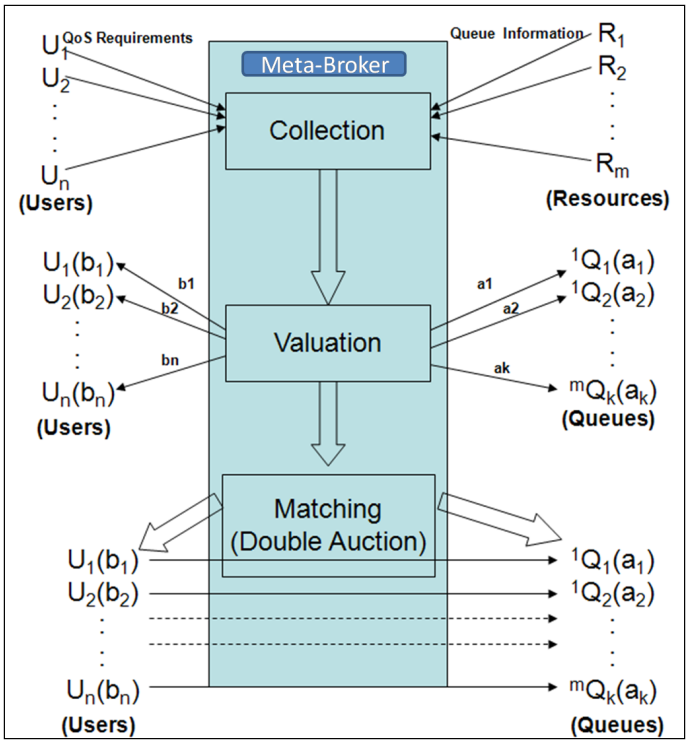


Figure 7.1: Double Auction based Meta-Scheduling Protocol

The elements of our meta-broker, which acts as an auctioneer in this context, can be divided into three parts, as shown in Figure 7.1 – *collection*: meta-broker collects information about queue slot availability and waiting time, *valuation*: assigns values to the user applications and resource queues, and finally, *matching*: matching of applications to resources. Within the meta-broker, an application valuation is considered as a *bid* while a resource valuation is considered as an *ask*. In Figure 7.1, U_n represents the n^{th}user application, a_k and b_n represent the ask k for resource queue k and the bid corresponding to user n respectively, and ${}^mQ_k$ represents the resource queue k at the resource m.

In our mechanism, the meta-broker generates bids and asks using the initial valuations

of users and resources as input, and augmenting them with information about resource availability and user requirements. The valuation methods are described in the next section. At regular scheduling intervals, the meta-broker matches the applications (asks) to the resource queues (bids) if the deadline constraint of the application is satisfied. If an application cannot be matched, then it will be reconsidered in the next scheduling interval. Throughout this thesis, we refer to our Double auction-inspired mechanism as DAM.

7.3.1 Valuation Mechanism

One of the most important components of an auction is to assign valuation to user applications and resources. The Grid services may be valued based on the cost of infrastructure, and economic factors such as supply and demand. However, user needs and urgency, and simultaneously, efficient utilisation of Grid services must be reflected through valuation of user applications and resources. Therefore, the meta-broker must generate a metric for both the users and the resource providers that takes into account all these attributes. This valuation should be dynamic, that is, in each scheduling cycle; it should be updated based on various parameters, and the dynamic demand and supply of the system. To design a good valuation metric, we took inspiration from Multi-Attribute Utility Theory (MAUT) [20] [93]. MAUT gives a logical, consistent and tractable approach for quantifying an individual's preferences by consolidating them into a single objective. This allows comparison of many diverse measures via single value. This theory includes first the identification of attributes and desirability function for each attributes and, then aggregation of these desirability functions to a single scalar utility value. The details of our valuation metric are discussed in next section.

Resource Valuation (Ask): The valuation of resources is affected by attributes such as waiting time, load, initial valuation of the provider, and economic factors such as demand and supply. The load of a queue is defined as the ratio of the number of processors occupied to the total number of processors available to the queue. In order to balance load across independent Grid sites, the meta-broker tries to give preference to the least loaded queues while submitting applications. Also, the most urgent application must be matched to the fastest queue. Since, in a Call auction, the maximum bid is matched to minimum ask, the valuation metric of resource queues should be such that the queue with the least waiting time should have the least ask value. Moreover, the valuation metric should also take into account the initial valuation given by the resource provider, and also the demand and supply levels of resources in the system (denoted as $Demand$ and $Supply$). $Demand$ is the sum of the processors required by tasks to be allocated and $Supply$ is the total number of processors available at all resources. Let $l_{k,t}$ be the load on the resource queue k at time t. Let $c_{k,t}$ be the initial resource valuation given by the provider. Let $w_{k,t}$ be

the application waiting time for queue k at time t. Thus, the desirability functions are proportional to $w_{k,t}$, $c_{k,t}$, $Demand$, $Supply$, and $l_{k,t}$. Since each of these attributes are preferentially independent [93], thus the valuation metric can be formed by aggregating these attributes either in multiplicative or additive form. When we compared the effects of each metric on distribution of load on the resource sites, we found that in the case of additive form, the deviation in load[1] across the resource sites is much more than that for the multiplicative form. This can be observed from Figure 7.2. Secondly, when more than two attributes are involved, an accurate additive aggregation requires the exact trade-offs between attributes, which may not be apparent. Thus, we have used a multiplicative aggregation to form a valuation metric from normalised attributes. This is given by:

$$a_k(t) = O_k \times w_{k,t} \times \frac{c_{k,t}}{c_{max,t}} \times \frac{l_{k,t}}{l_{max,t}} \times \frac{Demand}{Supply}, \text{where } O_k \text{ is a proportionality constant} \tag{7.1}$$

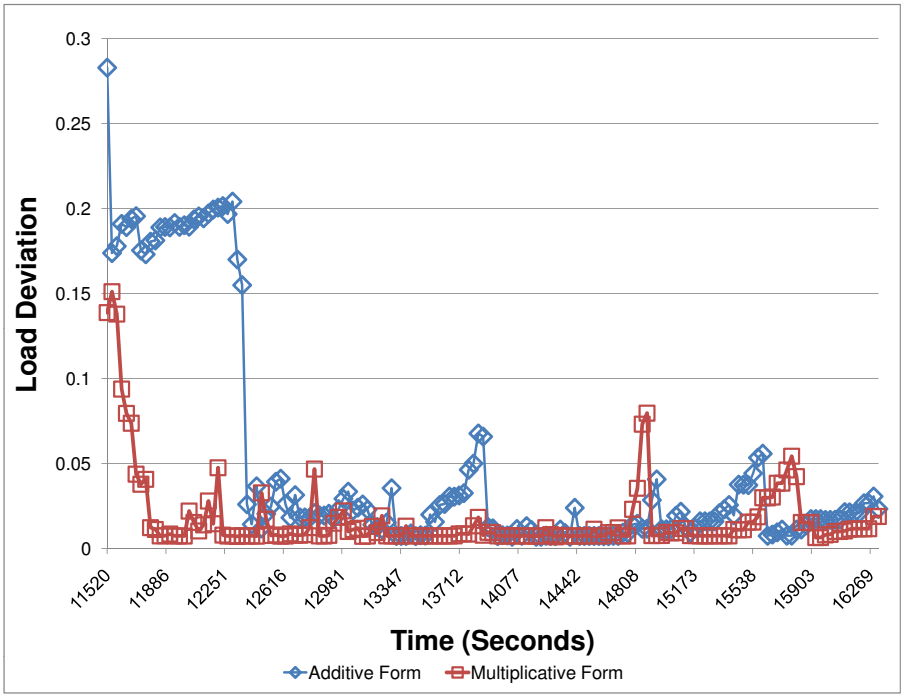


Figure 7.2: Comparison of Multiplicative and Additive forms of Valuation Metrics

Job Valuation (Bid): Similar to resources, a user's application has also many attributes such as the number of CPUs required, deadline and run time. The valuation metric of an application i at time t is designed in such a way that the maximum value is assigned to applications with urgent deadline. Also, if an application has not been mapped in the previous scheduling cycle, its corresponding bid (valuation) should be increased. This is to

[1]Note: Configurations of the experiment conducted is same as given in the Section 7.4.1

reduce the possibility of the application getting starved of resources by others with higher valuations (bids) that have arrived at the meta-broker in the meanwhile. The urgency can be calculated as $(d_i - t)$, where d_i is the user-supplied deadline for the application. Let st_i be the submission time of the application. Similar to that for resources, the application metric should also take into account the initial valuation of the application given by the user, and also demand and supply levels of resources in the system (denoted as $Demand$ and $Supply$). Let v_i is the initial application valuation given by the user. Thus, the desirability functions are proportional to v_i, st_i, $Demand$, $Supply$, and $(d_i - t)$. The resultant valuation metric, formed by multiplying all the normalised attributes, is the following:

$$b_i(t) = H_i \times \frac{v_i}{v_{max}} \times \frac{d_{max} - t}{d_i - t} \times \frac{Demand}{Supply} \times \frac{(t + 1 - st_i)}{t + 1 - st_min}, \tag{7.2}$$

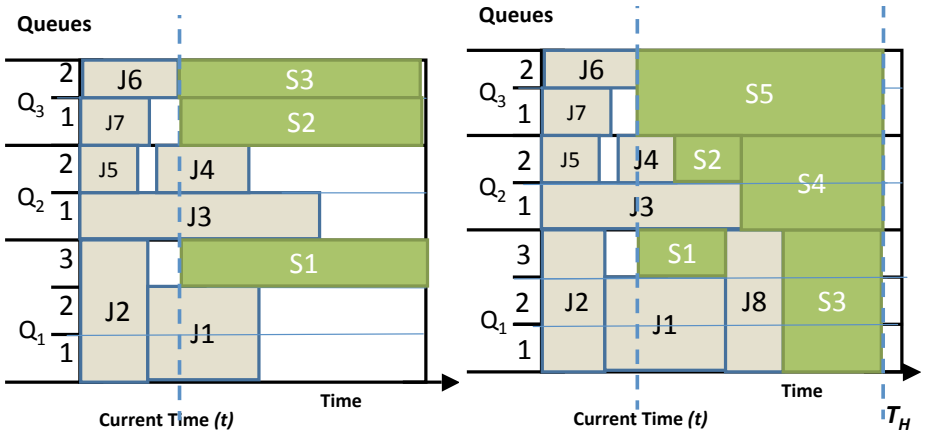
where H_i is a proportionality constant, and $d_i \neq t$.

7.3.2 The Meta-Scheduling Algorithm

As discussed earlier, the bids and asks generated by the meta-broker using the valuation metrics are sorted on the basis of their values. Let following be the ordering of asks and bids after sorting:

$$a_1 < a_2 < \ldots a_j \ldots < a_m$$
$$b_1 > b_2 > \ldots b_i \ldots > b_n$$

A user application is allowed to participate in the match process if $a_m < b_n$. As noted before in Section 7.2, the commodity unit matched on behalf of the resource site is a slot which a set of processors bounded by start and finish times. Figure 7.3 demonstrates the two methods by which the slots can be generated:



(a) Immediate (b) Packaged

Figure 7.3: Available Queue Slots

Immediate: At time t, the meta-broker can provide an *immediate* allocation to applications on the processors that are currently free. For example in Figure 7.3(a), two processors are available on queue Q_3. In this case, the slot start time is the scheduling instant, and the slot is allocated for the estimated run time of the application. In addition, the maximum number of slots available will equal the number of processors. This manner of slots trading can help in reducing fragmentation in the scheduling when application's runtime is not precise.

Packaged: Alternatively, the applications can be scheduled to fill up the queue to a specific time horizon T_H. The slots start from the first available time and contain as many processors that are not occupied for a specific duration. In Figure 7.3(b), the slots S_1 to S_5 are examples of such slots. After current time t, two processors are free in queue Q_3 upto time T_H. S_5 is therefore, an available time slot. Another time slot S_1, consisting of one processor on the queue Q_1, is available as well. In this method, the slot sizes can be of different sizes as can be noted from Figure 7.3(b). This approach, depicted in Figure 7.3(b), was also used by Singh et. al [147]. In this case, local schedulers can use backfilling to minimise the fragmentation in the schedule such that execution of applications in the queue does not get delayed.

Algorithm 7.1: Double Auction-inspired Meta-scheduling Mechanism

```
1  while current_time < next_schedule_time do
2      RecvResourcePublish(P)
       // P contains information about providers
3      RecvJobQoS(Q)
       // Q contains information about users
4  end
5  Calculate the Demand and Supply for resources
6  Update the value of bids and asks using eqn. 7.2 and 7.1
7  Sort asks in ascending order
8  Sort bids in descending order
9  while all applications are assigned to resource queues do
10     if bid b_i is greater than ask a_j then
11         if QueueWaitingTime(j) + ExecTime(i) < Deadline(i) then
12             if check processor availability on resource j then
13                 Schedule the application i to the resource j
14                 add application with matched resource site in Schedule List (Schd_List)
15                 update the value of available time slots with ask a_j
16                 i++
17         else
18             add user application to pending application list
19         end
20     j++
21 end
22 foreach element ∈ Schd_List do
23     notifyuser()
24 end
```

Our scheduling mechanism is shown in Algorithm 7.1. In each scheduling cycle, the meta-broker schedules the parallel applications after collecting all user's requests and resource performance information such as queue waiting times and free time slots (Line 1-3). At the end of each scheduling cycle, the meta-broker computes the demand for resources and their supply (Line 4). Then it assigns valuation to user applications (bids) and resource's queue (asks) using the mechanisms presented in the previous section (Line 5).

From the sorted bid list, the bid (b_i) with the highest value is matched to the resource queue with the minimum ask (a_j) that can satisfy the user requirement. Whether the user application i is scheduled to the resource queue j (corresponding to ask a_j), it will depend on the applications' processor and deadline requirements. Thus, first the deadline of application i is checked using waiting time of the resource queue j (Line 10) and then processor availability is checked on resource queue j (Line 11). If there is an ask which satisfies the application's QoS requirements, then the bid is matched to ask, and the user and the resource provider are informed of the match. The application i is then scheduled on to the resource queue j (Line 12) and then added to schedule list (Line 13). The available time slots (commodity units) on resource queue j are updated correspondingly (Line 14). If the deadline requirement of application i can be satisfied by resource queue j, then no other ask can be matched to the application's bid (Line 18). Therefore, the application is removed from the bids list in that scheduling cycle. If the required number of processor is not available on the resource queue j, bid b_i is matched with next ask in the sequence (Line 19). Matching for other bids and asks will be repeated until either all bids or asks are matched.

7.3.3 Queueing Theory Based Model for Meta-scheduling

In order to know the generalised behaviour of any meta-scheduling algorithm, it is important to analyse it with a queuing model which will also help to test the near optimality of the algorithm. Thus, we designed a formal mathematical model to analyse and predict the performance of the DAM in terms of other system-based metrics such as slowdown and waiting time. This analytical model which finds the expected (average) metrics for our meta-scheduling heuristic is used to test the correctness and substantiate the strength of DAM in the experimental section.

Our analytical model is based on queuing theory, which has been used extensively for modelling scheduling problems in distributed systems. Queuing Theory provides a powerful stochastic and probabilistic approach to analytically model the mechanisms of scheduling policies [100]. It can be observed by the queueing view of the system model considered in previous section (as shown in Figure 7.4) that the system is well suited to be analysed via queueing theory.

Using this model we can get expected performance metrics such as mean waiting time and mean slowdown of applications, which can be used to directly compare the performance of our proposed meta-scheduling mechanism. An application's slowdown is its waiting time divided by its execution time. Mean slowdown is considered because users generally desire that their application delay should be proportional to its execution time [135][76]. For instance, a user with small application size generally prefers to wait for relatively lesser time than those users who have longer applications.

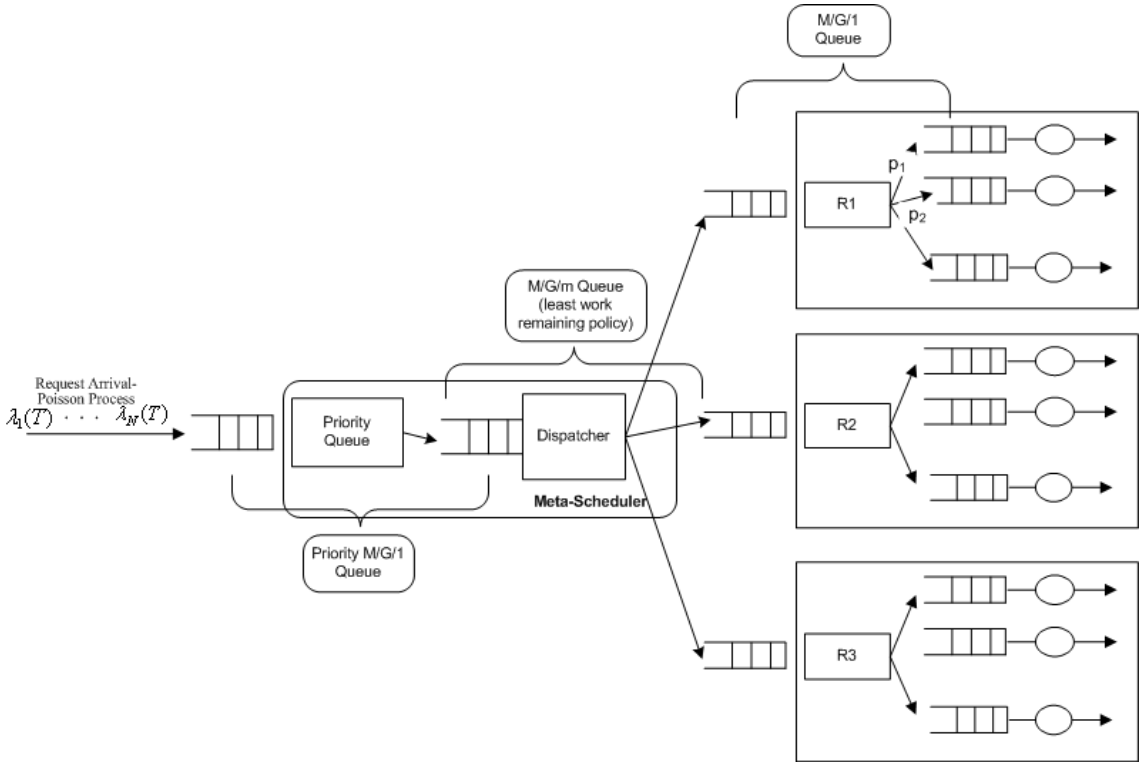


Figure 7.4: Queuing Theory View of Meta-scheduling

We can model the system under consideration as a network of three queues to get the optimal bound for various system parameters. In the meta-scheduler, each application is assigned a priority or valuation and matched to resources according to its priority. This component of meta-scheduler can be modelled by a priority queue system. Since processing time of each application may not be exponential, thus we have chosen M/G/1 priority queue system to analyse this part of the meta-scheduler. Then, an application is assigned and dispatched to a resource to balance load across all queues in the system. This component of the meta-scheduler is analysed using a comparable queueing system i.e. central queueing system with least work remaining policy (M/G/m).

At each resource, an application is executed on more than one machine at the same time. The local scheduler of the resource site also uses the backfilling policies and different types of queues to increase utilisation [115]. This component of meta-scheduling

is difficult to model analytically. Thus, to analyse this part of meta-scheduling, we have taken an approximate model of real local scheduling systems. We have divided CPUs of a resource into multiple partitions, where each partition acts as a M/G/1 queueing system. The resource with multiple partitions is an approximation of real local scheduling systems. The CPU/machine processing requirements of each application also follows a general distribution (denoted as G).

Thus, in meta-scheduling, each application goes through three queuing systems before it start executing. Hence, by analysing the combination of the following sequential network of three queuing systems, we get an approximate analytical model for meta-scheduling system:

- Valuation or prioritisation (M/G/1 Queue with Priority)
- Matching or Dispatching (M/G/m Queue i.e. least work remaining (LWR) policy)
- Scheduling at a resource for execution (M/G/1 Queues)

The mean arrival rate of applications is λ. Let the service requirement distribution of applications be any general distribution X. The mean service time of the applications is $E(X)$ and the second moment of service time is σ or E(X^2). The distribution of processor requirement by each application is given by C. Let the number of resource sites be m.

Valuation or prioritisation (M/G/1 Queue with Priority)

In the first part of DAM, applications are assigned valuations so that they can be re-sequenced for scheduling. Thus, this scenario can be modelled as a single server queuing system as shown in Figure 7.4. There are K priority classes of applications. The mean service time of priority class j is $\frac{1}{\mu_j}$ and the second moment of service time is $\sigma_j^{'}$. The overall second moment of service rate of applications by the server is $\sigma^{'}$. Since the time taken to assign priorities is very small in DAM mechanism, the mean service rates of application will also be very small. The applications arrival follows a Poisson distribution with mean λ.

Let p_j be the probability with which j^{th} priority class applications arrived at the server. Then, the mean arrival rate (λ_j) of the j^{th} priority applications is given by:

$$\lambda_j = p_j \times \lambda \tag{7.3}$$

Since the service time distribution of all priority class is $\frac{1}{\mu_j}$, thus the system load due to j^{th} priority class applications is given by:

$$\rho_j = \frac{\lambda_j}{\mu_j}$$

Let $E(w_j)$ is expected waiting time for the applications with priority level j. Then using the classic result on non-preemptive priority queue by A. Cobham [38] we obtain the mean waiting time and slowdown ($E(s_j)$) of class-j applications:

$$E(w_j) = \frac{\frac{\lambda \times \sigma^{'}}{2}}{(1 - \sum_{i=1}^{j-1} \rho_i)(1 - \sum_{i=1}^{j} \rho_i)}$$

Thus, total mean waiting time and slowdown of applications in the system is given by:

$$\bar{W_1} = \sum_{j=1}^{K} (p_j \times (E(w_j) + \frac{1}{\mu_j})) \tag{7.4}$$

$$\bar{S_1} = \sum_{j=1}^{K} (p_j \times (E(w_j) + \frac{1}{\mu_j})) \times E(X_j^{-1}) \tag{7.5}$$

Matching or Dispatching (M/G/m Queue)

After applications are served by above queuing system based on priority, applications in the out-going queue will be served by the second queuing server on a FCFS basis. It can be considered as third component of meta-scheduler i.e., matching. For simplicity, the applications arriving into the second queueing system are taken to follow a poisson process. For the assignment of these applications to the resource sites, we have used the central queue with m servers policy. This policy has been proven to be equivalent to least-work-remaining allocation policy, which is claimed to be optimal by Nelson et. al [119][120]. This policy is not analytically tractable under M/G/m queueing system. Nonetheless, several good approximations exist in the literature, many of which are empirical. In this study, we use the approximation given by Nozaki et. al [123] which is used in several other publications [75] [76]. The approximation for mean queue length is stated as:

$$E(N_{M/G/m}) = E(N_{M/M/m}) \frac{E(X^2)}{E(X)^2}, \text{where X: Service Requirement \& N: Queue Length} \tag{7.6}$$

The load of system is given by:

$$\rho = \lambda \times E(X)$$

Let $E(W_{M/M/m})$ be the average waiting time in a M/M/m queueing system and ρ be the system load. Then, using well known Pollaczek-Khinchin formula in queuing theory,

the average queue length and waiting time for M/M/m queueing system is given by:

$$E(N_{M/M/m}) = \frac{P_N \rho}{1-\rho}, \text{where } P_N = [\sum_{z}^{m-1} \frac{(m\rho)^z}{z!} + \frac{m^m \rho^m}{m!1-\rho}]^{-1} \tag{7.7}$$

$$E(W_{M/M/m}) = \frac{E(N_{M/M/m})}{\lambda} \tag{7.8}$$

Thus, using Equation 7.6, 7.7 and 7.8, the mean waiting time and slowdown in the queue by using central queue policy is given by:

$$\bar{W_2} = E(W_{M/G/m}) = E(W_{M/M/m}) \frac{\sigma}{E(X)^2} \tag{7.9}$$

$$\bar{S_2} = \bar{W_2} \times E(X^{-1}) \tag{7.10}$$

Scheduling at a Resource for Execution (M/G/1 Queues)

After the application is assigned to the resource queue, the application is needed to be scheduled on multiple servers. Unlike the most of the commonly used queuing systems where an application requires only one server for execution, here each application needs more than one server (processors in our context) at the same time. Moreover, local scheduler at a resource uses different backfilling policies to decrease slowdown of applications [115]. Since, it is analytically intractable to solve this system, the designed analytical system for this component of meta-scheduling differ slightly from real systems. We divided the processors of the resource into multiple disjoint partitions, with one queue per partition. Each partition f on resource z is initially assigned r_{zf} processors. Each of these queues processes the applications, which require processors within range of $(r_{z(f-1)}, r_{zf})$, on FCFS basis. Let there be N_z servers/processors at resource site z, which are divided between n_z queues. Thus,

$$r_{z0} + r_{z1} + r_{z2} + r_{z3} ... r_{zf} + .. r_{z(n_z-1)} = N_z \tag{7.11}$$

Since the total number of resource sites is m, the arrival rate of applications at resource site z is given by:

$$\lambda_z = \frac{\lambda}{m} \tag{7.12}$$

Let u_{zf} be the probability that an application require processor between $r_{z(f-1)}$ and r_{zf}, and thus processed by queue f of resource site f. Let C be the probability distribution of processor requirements for an application, this probability is given by:

$$u_{zf} = \int_{a_{z(f-1)}}^{a_{zf}} C(x)dx \tag{7.13}$$

Thus, the fraction of applications arriving at queue f of the resource site z is given by:

$$\lambda_{zf} = \lambda_z u_{zf} \tag{7.14}$$

The load shared by each queue, when $E(X)$ is mean service time, is given by:

$$\rho_{zf} = \lambda_{zf} E(X) \tag{7.15}$$

Since each queue partition has M/G/1 FCFS queue system behaviour, thus we can directly use the same results for the average waiting time. This is given by:

$$E(w_{zf}) = \frac{\lambda_{zf}\sigma}{2(1-\rho_{zf})} \tag{7.16}$$

The total expected waiting time at a resource site is the average of all waiting time at each queue partition, which is given by:

$$E(w_z) = \frac{1}{n_z}\sum_{f=1}^{n_z} E(w_{zf}) \tag{7.17}$$

Let W_3 and S_3 be the expected waiting time and slowdown of all resource sites, respectively. Thus, they are given by:

$$W_3 = \frac{\lambda}{m}\sum_{z=1}^{m} E(w_z) \tag{7.18}$$

$$S_3 = W_3 \times E(X^{-1}) \tag{7.19}$$

The overall expected waiting time and slowdown measures are given by combining waiting time and slowdown of all three queueing system, i.e., Equation 7.4-7.5, 7.9-7.10 and 7.18-7.19:

$$\text{Waiting Time} = E(W) = W_1 + W_2 + W_3 \tag{7.20}$$

$$\text{Slowdown} = E(S) = S_1 + S_2 + S_3 \tag{7.21}$$

Thus, the queueing theory based analytical model for our meta-scheduling mechanism is given by following equations:

$$Minimise(E(W)) \text{ subject to } \sum_{k}^{n_z} r_{zk} = N_z, 1 < z < m \tag{7.22}$$

$$Minimise(E(S)) \text{ subject to } \sum_{k}^{n_z} r_{zk} = N_z, 1 < z < m \tag{7.23}$$

The above model gives an approximation for real meta-scheduling systems. Thus, to predict performance of our meta-scheduling policy, we can obtain optimal expected waiting time and slowdown. Thus, Equations 7.22 and 7.23 for expected waiting time and slowdown are needed to be solved for different values of n_z using optimisation tools such as Mathematica [Wolform Research 2008]. These analytical values are used to ascertain the performance of DAM.

7.4 Performance Evaluation

7.4.1 Experimental Configuration

For our experiments in this chapter, we use Feitelson's Parallel Workload Archive (PWA) to model the parallel application workload for Grids. Since this chapter focuses on studying the HPC parallel applications of users, the PWA meets our objective by providing the necessary characteristics of real parallel applications collected from supercomputing centers. To avoid affect of initial setup phase of the HPC center, our experiments utilise the applications in the second week of the Lawrence Livermore National Laboratory (LLNL) Thunder (January 2007 to June 2007). The LLNL Thunder trace from the LLNL in USA is chosen due to its highest resource utilisation of 87.6% among available traces to ideally model a heavy workload scenario. From this trace, we obtain the submit time, requested number of processors, and actual run time of applications. The characteristics of the traces are given in Table 7.1. The submission time of parallel application is divided by 1000 to increase the number of applications submitted per schedule interval as per the methodology presented by Sanjay and Vadhiyar [141]. Since the workload trace does not contain any information about the user's deadline and initial valuation, these were generated synthetically. For a user application with a runtime r, the deadline was generated from a uniform random distribution between r and $3r$. The trace data of utility Grid applications are currently not released and shared by any commercial utility Grid providers, thus this information also has to be generated using a random distribution. The average initial valuation of applications is chosen randomly between 90000 and 160000 currency units using uniform distribution, so that it is always greater than application execution cost. The user valuations are assigned so that at least half of users can afford to execute

their application on the resources with the highest valuation. The Grid modelled in our

Table 7.1: Workload Characteristics

Mean Inter-Arrival Time	16.176 sec.
Average Job Runtime	2328.911 sec.
Standard Deviation for Job Runtime	7790.11 sec.
Average CPU requirement	44 CPUs

simulation contains 10 resource sites spread across five countries derived from European Data Grid (EDG) testbed [83]. The configurations assigned to the resources in the testbed for the simulation are listed in Table 7.2. The configuration of each resource is decided so that the modelled testbed would reflect the heterogeneity of platforms and capabilities that is normally the characteristic of such installations. Each of the resources is simulated using GridSim [24] as a cluster that employed a multi-partition easy backfilling policy for local resource allocation [44].

The processors associated with each cluster in Table 1 are exclusively managed by the meta-broker (i.e. all users are going through meta-broker). We have sub-divided the allocated CPUs of each cluster into 3 queues in ratio of 1:2:3 of the total number of CPUs in the cluster. The processing capabilities of the processors are rated in terms of Million Instructions per sec (MIPS) so that the application requirements can be modelled in Million Instructions (MI). The average initial valuations that are assigned to each resource is between 4.5 and 9.5 currency units per processor per second.

We have compared our Double auction-inspired meta-scheduling mechanism against five other well-known traditional and market based mechanisms listed below:

- **Shortest Job First (SJF):** In this mechanism, the applications are prioritised on the basis of estimated runtime. This is a very common algorithm used in cluster management.

- **First Come First Serve (FCFS):** An application is assigned to the first available queue. This is a common mechanism employed by many meta-schedulers such as GridWay [85].

- **Earliest Deadline First (EDF-FQ):** In this mechanism, the applications with the earliest deadline are scheduled on to the resource queue slot with the least waiting time.

- **Highest Valuation to Fastest Queue (HVFQ):** In this mechanism, the application with the highest user valuation is assigned to the queue slot with the least waiting

Table 7.2: Simulated EDG Testbed Resources

Site name (location)	**Number of processors**	**Single processor rating (MIPS)**
RAL (UK)	2050	1140
Imperial College (UK)	2600	1330
NorduGrid (Norway)	650	1176
NIKHEF (Netherlands)	540	1166
Lyon (France)	600	1320
Milano (Italy)	350	1000
Torina(Italy)	200	1330
Catania (Italy)	250	1200
Padova (Italy)	650	1000
Bologna (Italy)	1000	1140

time. This mechanism is generally used in auction mechanism such as Vickrey auction. Vickrey auction is used in resource management systems such as Spawn [165] and Bellagio [9].

- **FairShare or Proportional Share:** In this mechanism, each application is assigned queue slots proportional to the ratio of its valuation to the combined valuation of all the applications. This mechanisms is employed in REXEC [36].

The following criteria are used to compare fairness and user satisfaction provided by these mechanisms:

- **Urgency vs. Success Ratio:** The user's urgency to get their application completed, is defined as:
$$u = \frac{deadline - start\_time}{execution\_time} - 1 \tag{7.24}$$
where $start\_time$ and $execution\_time$ are attributes of the application. The deadline is considered very urgent when $u < 0.25$, urgent when $0.25 < u < 0.5$, intermediate when $0.5 < u < 0.75$, relaxed when $1 > u > 0.75$ and very relaxed when $u > 1$. This criterion relates to how the scheduler deals with users with different demands on time.

- **Valuation vs. Success Ratio**: The valuation provided by the user for an application is divided by the required number of processors in order to normalise it. We examine how the schedulers allocate resources fairly among different users with different

application valuations. If $(u < 0)$ for an application then the application will not be scheduled by the meta-broker.

- **Number of deadlines missed** with increase in number of user applications. We use this criterion to examine how the scheduling mechanisms are able to cope with user requests when demand for resources exceeds supply.

- **Load Deviation:** The load of a resource is the ratio of the number of processors occupied to total number of processors available at the resource site. We average the load over the Grid resources and measure the standard deviation. This informs about how well the scheduling mechanism is able to balance load across the Grid.

7.4.2 Analysis of Results

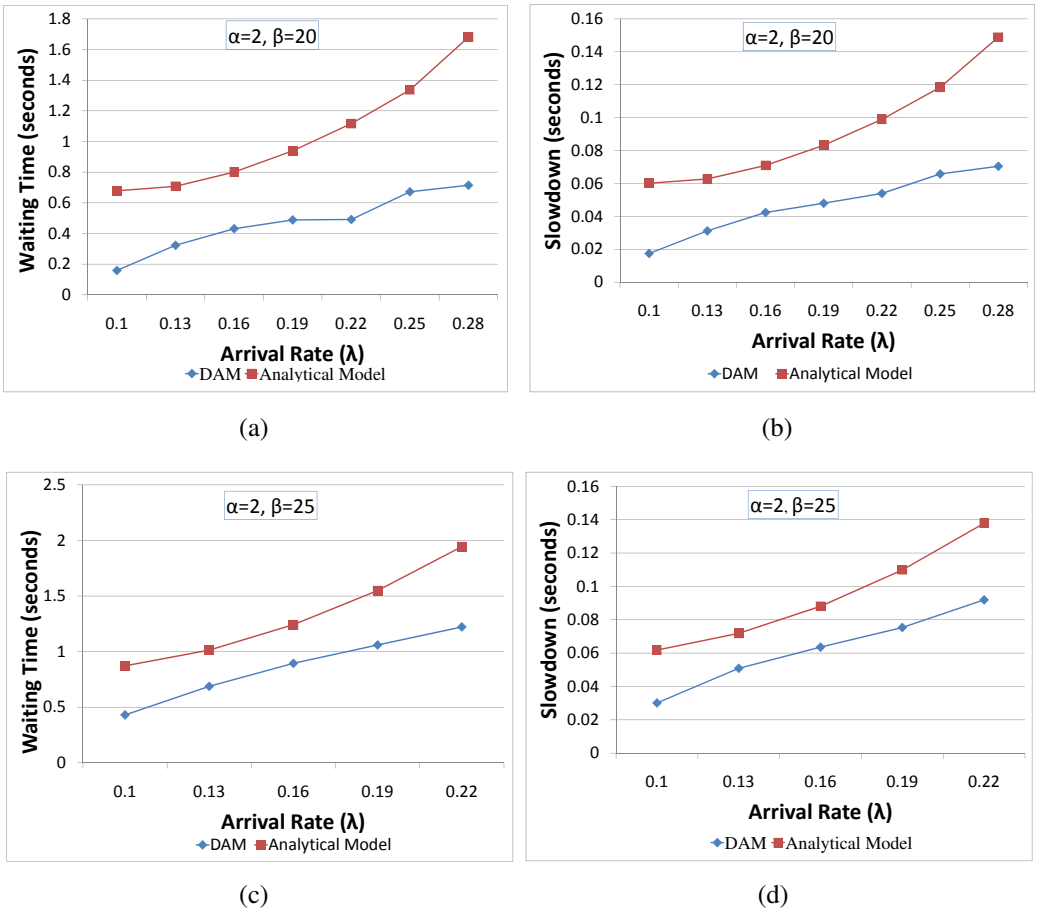


Figure 7.5: Comparison of DAM with analytical results

In this section, we discuss the results of our evaluation.

Comparison of DAM with Theoretical Results

In Section 7.3.3, we designed an analytical model for meta-scheduling problem in order to ascertain the behaviour and substantiate the strength of our proposed Double auction-inspired meta-scheduling (DAM) mechanisms. The analytical model gives an near-optimal solution for crucial system metrics such as mean waiting time and mean slowdown of applications, that can be compared with DAM and evaluated under a variety of conditions.

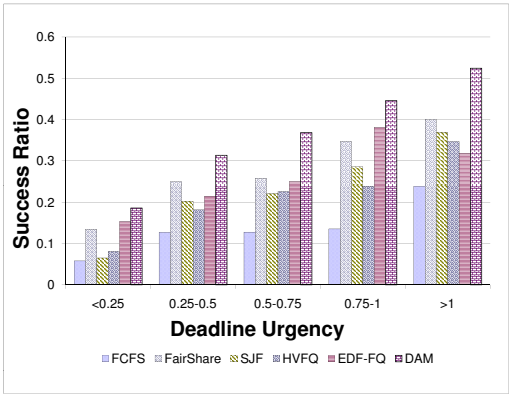
The optimal mean waiting time and mean slowdown is calculated using the approximate queuing model for meta-scheduling, by solving Equation 7.22 and 7.23 using the NMinimize function in Mathematica [Wolform Research, 2008]. This is achieved by finding the r_i values in each instance that produce the local minima for expected waiting time (E(W)) and slowdown (E(S)). Since the analytical model proposed in Section 7.3.3 is an approximation to the meta-scheduling problem, thus the optimal solution obtained is actually a near-optimal solution for the meta-scheduling problem at steady state.

Experimental Methodology: Li et al. [106] analysed various Grid Workloads and found that the Weibull distribution is best fitted to model runtime of applications. Thus, the application runtime is generated using a Weibull distribution (α, β) [106]. The arrival rate application is assumed to be Poisson distribution with parameter λ. Since our aim is to compare both analytical and simulation results, thus the probability distributions for arrival rate and runtime of applications are same in both analytical and simulation experiments. However, since the analytical model is an approximation of the real systems, thus the r_i values can differ between the analytical model (where they are numerically solved) and the simulated real systems (where the scheduling is done using heuristics). The arrival rate of i^{th} priority class applications (λ_i) depends on probability p_i. The p_i is obtained during the valuation process of DAM through simulation. Each resource considered is assumed to have same number of processors to make the simulation scenario as close as possible to the analytical model. To make the solution of the analytical model tractable, we have considered five Grid resources with 128 processors each and four priority class applications.

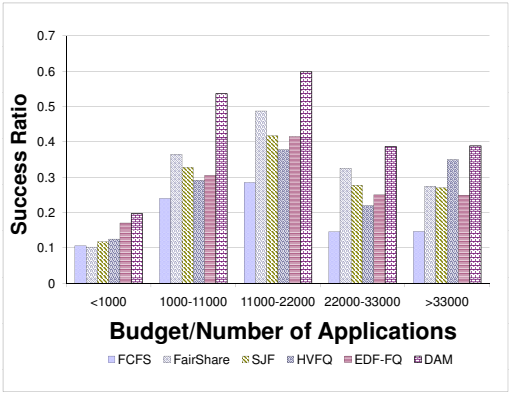
A large range of λ values were considered demonstrating a wide spectrum of load and arrival rate. The performance metrics were computed for these arrival rates each with different mean application runtime (represented by the combination of values of α and β). Two set of results with different mean application runtime are presented in Figure 7.5. The different mean in two scenarios is obtained by scaling up the value of β which results in jobs with longer average runtime and with large variance.

In order to obtain steady state results, we ignored scheduling of first 5000 applications and measure mean waiting time and mean slowdown for next 5000 applications. For each value of λ, experiment is repeated 30 times and average of results from these repeated experiments is used for comparison.

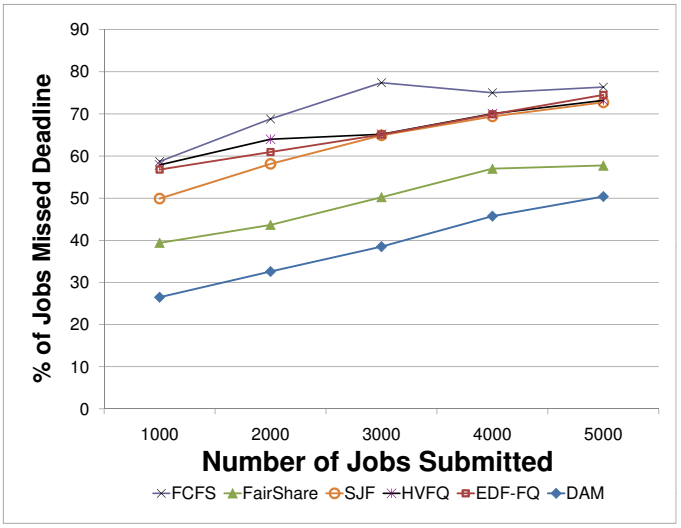
Discussion of Results: Figure 7.5 shows that the simulation results of DAM follows similar increasing trend as the analytical model. This not only validates the correctness of DAM heuristic but also indicates that the performance of DAM is near optimal in terms of metrics such as mean waiting time and mean slowdown. DAM gives consistently lower values for waiting time and slowdown than the optimal values obtained from analytical model. There is significant gap between DAM and analytical model results, for example, when $\beta = 25$, the gap between DAM and analytical model is about 25% to 30%. The reason for the gap is that the analytical model is an approximation of real meta-scheduling systems and thus it does not model the backfilling policies used by the local scheduler of resources which reduces the slowdown and waiting time of applications more than the optimal solution of analytical model.



(a) Effect of user urgency



(b) Effect of user valuation



(c) Number of Deadlines Missed

Figure 7.6: Benefit for Users

Benefit for Users

This section shows how our meta-broker benefits users by not only completing the most number of applications with different QoS needs but also benefiting every user in different urgency and budget groups.

Effect of User Urgency: Figure 7.6(a) shows the percentage of total applications completed successfully against the users' urgency values. Figure 7.6(a) shows that DAM has scheduled a larger number of applications than other mechanisms in every urgency group. For example, in the intermediate group $(0.5 - 0.75)$, DAM scheduled 12% more applications than its closest competitor (FairShare). This is in contrast to the performance of FCFS and SJF which is the worst in almost every case. This is due to the fact that DAM is designed to increase an application's value with urgency, while in others this is not considered. The performance of FairShare is very close to DAM and even scheduled almost equal number of applications as DAM when Deadline urgency is less than 0.25. This is because DAM tries to reduce the waiting time of applications with relaxed deadline by increasing their valuation. Thus, when the deadline urgency was greater than 1, then DAM scheduled about 12% of more applications than FairShare. Jobs with relaxed deadlines progressively gain in valuation (or, float to the top of the bid list) when they are held at the scheduler over time in DAM, and are therefore not starved. This can be seen by comparing the performance of DAM with EDF-FQ, which prioritises urgent applications but performs poorly with relaxed deadlines. Since the users' objective is to complete their applications by the deadline, delaying an application at the scheduler is appropriate as long as the deadline is met.

Effect of User Valuation: From Figure 7.6(b), we can see that DAM completes greater number of applications across all valuations than the other mechanisms. Even though Fareshare again performed very close to DAM for medium valuation groups, DAM outperform FairShare for all other groups by scheduling atleast 10% more applications. For applications with very low valuation (< 1000), the DAM managed to schedule about 20% of the applications as compared to 11% for FairShare which perform as well as DAM when the application valuations are medium. This is because the latter assigns the lowest proportion of resources to the users with the lowest valuation. Therefore, in this case, most of the parallel applications fail to execute due to lack of sufficient processors. It is also interesting to note that HVFQ, which was supposed to favour users with high budget, has scheduled almost 4% less number of applications than DAM for $Budget > 33000$. This is because HVFQ does not consider other requirements of applications such as deadline.

Number of deadline missed: From Figure 7.6(c), we can clearly see as the demand for resources (number of applications) increases, the number of applications that missed their deadline also correspondingly increases due to the scarcity of resources. But DAM is able to complete around 8% to 15% more applications than other mechanisms. As SJF resulted in better packing of jobs at resource sites, SJF performed relatively better than the other mechanisms such as EDF-FQ, HVFQ and FCFS. FCFS performs the worst as it does not consider the effect of deadlines and queue waiting time.

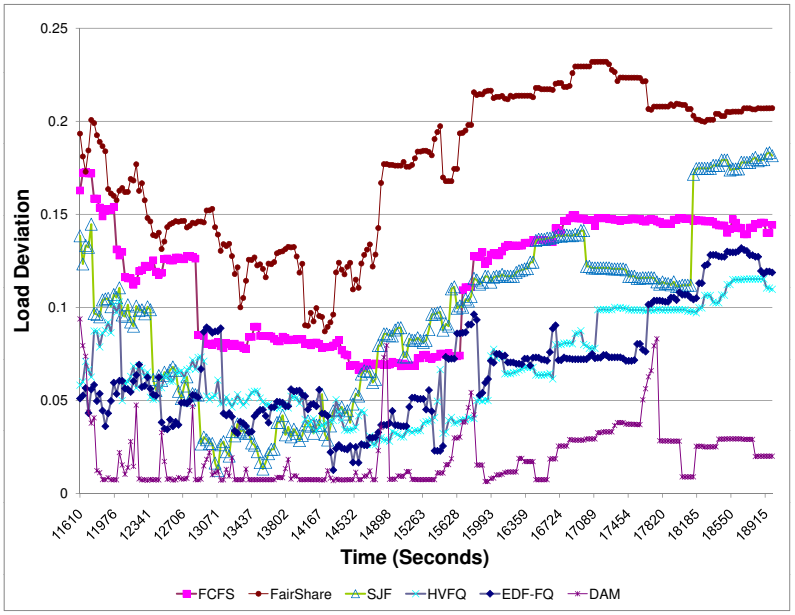


Figure 7.7: Variation in Load across Resources

Benefit for Resources

Simulation results in Figure 7.7 show how DAM effects the load on different resources. The figure shows the standard deviation in resource loads against the time period of the execution. It can be noted that while the deviation across resources for other mechanisms are steadily increasing, on average DAM has kept it, consistently, almost close to 0. This implies that the DAM was able to successfully balance the load across all the resource sites. This is due to the fact that the resource queue's valuation is increased when its load is increased and therefore, heavily loaded queues are sorted to the bottom of the ask list. The performance of the EDF-FQ mechanism, which is closest to that of DAM, also resulted in on average 5 times more Load Deviation than DAM. Moreover, from Figures 7.6(a) and 7.6(b), it can be seen that EDF-FQ does not schedule as many applications as DAM, even though EDF-FQ also tries to balance load across the resources by submitting according to queue waiting time. However, FairShare mechanism which benefitted the users in similar way as DAM, resulted in maximum load imbalance which is even worst than HVFQ.

Thus, DAM is not only providing benefit to users but also providing benefit to resource providers by equally dividing load between them.

7.5 Summary

We have presented a market-oriented meta-scheduling policy for allocating multiple applications on distributed resources within a Grid. The resource sites are organised as a collection of queues with different capabilities and waiting times. The goal of the scheduler is *to maximise the users' utility* by taking into account their deadlines and the value attached to their applications, *and to maximise the resource providers' utility* by fairly allocating the applications across the Grid. The meta-broker also has to benefit all users by preventing starvation of applications that have relaxed deadlines or low valuation, while keeping effect on other crucial system metrics such as mean waiting time and slowdown at minimum level.

Experimental evaluation of the proposed mechanism against common mechanisms, such as SJF, HVFQ, EDF-FQ, FCFS, and FairShare, used in other meta-schedulers has showed that DAM maximises the utility of both users and resource providers across all the target metrics. DAM is not only able to schedule about 8% to 15% more user applications, but also has the highest success ratio in almost all the groups for applications with different deadlines and different valuations. For the users with the lowest budget (< 1000), the success ratio of their applications is increased by almost 10%. Similarly, DAM also maximises the utility of resource side by equally distributing the workload according to capacity of resource sites. Thus, DAM is able to improve the balancing of load across the constituent resources of the Grid with almost zero load deviation. In maximising providers' utility, DAM even outperforms the FairShare mechanism which provides similar utility to users as DAM. The key reason here is that the valuation metrics are able to capture information which is important to both users and resource providers, and therefore the meta-broker schedules applications more effectively.

Thus, we demonstrate that, by considering both system metrics and market parameters, we can enable more effective scheduling which maximises both users' and resource providers' utility. In the next chapter, we will describe how the meta-broker can be implemented within a Grid exchange environment.

Chapter 8

Market Exchange and Meta-Broker Implementation

This chapter presents the implementation of our meta-broker within a market exchange (named as 'Mandi') framework. The resulting system aims to provide a market-oriented meta-scheduling environment in which users can trade and reserve resources in utility Grids. The architecture of Mandi is designed to overcome the limitations of the existing Grid Market infrastructures [4] [121], which were discussed in Chapter 2. The Mandi software also serves as a proof of concept for this thesis. In the next section, the motivation for designing Mandi is discussed. In the remaining sections of the chapter, market exchange's requirements, its architectural design, and the implementation of "Mandi" are discussed.

8.1 Motivation

Emerging utility computing paradigms such as Clouds are promising to deliver highly scalable HPC infrastructure on-demand. In Chapters 2 and 3, we discussed how a market exchange can ease the process of buying and selling of compute resources in these utility Grids. A market exchange provides a shared trading infrastructure which can enable interaction between different market-oriented systems.

As described in Chapter 3, our proposed meta-broker architecture distinguishes itself from other meta-scheduling systems by separating the job submission and monitoring (performed by the user brokers) from the resource allocation (performed by the meta-broker). The existing market exchange systems, such as Catnet [54], Bellagio [9], GridEcon [4] and SORMA [121], do not allow such separation, which makes them difficult to customise. In addition, they have been designed to support a limited number of trading protocols. This restricts the participation of resource providers using a trading protocol of

their own choice. The flexibility to choose trading and pricing protocols is critical since it can impact the participants utility enormously depending on the current demand and supply. Thus, a market exchange requires to support diverse services [16] including (a) registration, buying and selling; (b) advertisement of free resources; (c) coexistence of multiple market models or negotiation protocols such as auction; and (d) Grid resource brokers and RMSs to discover resources/services and their attributes (e.g., access price and usage constraints) that meet user QoS requirements.

Therefore, we introduce the design of a market exchange framework 'Mandi', which supports the above features, and thus overcomes some of the short-comings of the existing systems, by allowing the co-existence of multiple negotiations of different types. We aim to develop a light weight and platform independent service-oriented market architecture whose features can be accessed easily by current Grid systems. This chapter describes the implementation of our meta-broker architecture (proposed in Chapter 3) within Mandi market exchange, and how it enables the reservation of resources from multiple sites.

8.2 Market Exchange Requirements

The market exchange framework requirements can be divided into two categories: infrastructure and market requirements.

8.2.1 Infrastructure Requirements

(a) **Scalability:** Since, the increase in the number of resource requests can affect the performance of the market exchange, thus the scalability of the exchange can become an issue. The exchange architecture should be designed such that access to market services be least effected by the number of service requests. In addition, it should guarantee the best efficiency in matching the consumers demand and provider's supply.

(b) **Interface Requirements and Grid Heterogeneity:** The user interface plays an important role in allowing the access of the market exchange by a wide variety of users. Depending on how a user wants to access the market, different types of interfaces should be provided. In Grids, many market based brokers [85] [163] ease the process of accessing the Cloud resources. Similarly, on the resource provider side, heterogeneous resource brokers [104] with market based capabilities are available. Thus, these brokers should seamlessly access the market exchange's services whenever required by invoking simple platform independent exchange APIs.

(c) **Fault Tolerance:** the market exchange should be able to resume its services from the closest point before the failure.

(d) **Security:** To avoid spamming, there should be a security system for user registration. All the services of the exchange must be accessed by authorised users.

8.2.2 Market Requirements

(a) **Multiple types of Application Models and Compute Services:** The user resource requirements can vary according to their application model. For example, to run an MPI application, users may want to lease all the compute resources from same resource provider, which gives much better bandwidth for communicating processes. Thus, users can have different types of compute resource demands depending on their applications. Similarly, resource providers can advertise different type of resources such as storage and virtual machines. Thus, the market exchange should be generic enough to allow the submission of different types of compute resource requests and services.

(b) **Multiple User Objectives:** Users may wish to satisfy different objectives at the same time. Some possible objectives include receiving the results in the minimum possible time or within a set deadline, reducing the amount of data transfer and duplication, or ensuring minimum expense for an execution or minimum usage of allocated quota of resources. Different tasks within an application may be associated with different objectives and different QoS requirements. The exchange should, therefore, ensure that different matching strategies meeting different objectives can be employed whenever required.

(c) **Resource Discovery:** Users may have different resource requirements depending on their application model and QoS needs. Thus, the exchange should be able to aggregate different compute resources and should allow users to access and discover them on demand.

(d) **Support for Different Market Models:** In Grids, several market based mechanisms have been proposed for trading resources using market models such as auctions and commodity market [5]. Each mechanism, such as the English auction and the Vickery auction, has different matching and pricing strategies and has their own advantages and disadvantages. Thus, the exchange should be generic enough to support as many market models as possible.

(e) **Coexistence/Isolation of Market Models:** Similar to real world markets, the market exchange should support concurrent trading of compute services by different negotiation protocols such as auction. For example, Double auction and Vickery auction can coexist simultaneously and users can participate in each of them.

(f) **Support for Holding, Joining and Discovery of Auctions:** Users can have requirements that may not be fulfilled by currently available compute resources, and thus, may want to hold their own auctions and invite bids. Moreover, any user can discover these auctions and join them if necessary.

The following sections present the architecture, design, and implementation of Mandi market exchange that takes into account the challenges mentioned so far, and abstracts the heterogeneity of the environment at all levels from the end-user.

8.3 Mandi Architecture and Design

8.3.1 Design Considerations and Solutions

The primary aim of Mandi is to provide a marketplace where the consumer's resource requests and the provider's compute resources can be aggregated, and matched using different market models. The main challenges and the way they are approached in Mandi design are as following:

(a) **Flexibility in choosing market model and user objectives:** As discussed various market models or negotiation protocols can be employed by users to trade compute resources. Each market model has different requirements [5]. For example, in the commodity market model, consumers search the current state of the market and immediately buy some compute service. This requires synchronous access to that instance of compute resource. In the case of an auction, there is a clearing time when the winner selection is done. In addition, any user can request to hold auctions which require separation of each auction. Thus, the components within Mandi are designed to be modular and cleanly separated on the basis of functionality. Each of them talks through the persistence database that constitutes a synchronisation point in the whole system. Different auction protocols are abstracted as "one-sided auction" and "two-sided auction" which can be extended to add new auction mechanisms. Each auction type is characterised by the winner selection mechanism. The reservation of matched services is separated from the trading mechanisms to allow flexibility and coexistence of different trading models such as commodity and auction market.

(b) **Support heterogeneous compute resources and applications:** To allow heterogeneous Resource Management Systems (RMSs) and brokers to access market exchange services, Mandi's architecture is needed to be platform independent. Current market exchanges such as Sorma [121] handle the heterogeneity by implementing plug-in for each resource management system. This is not feasible in the long term

since APIs of different resource providers for job submission, reservation and monitoring may get updated with time. Thus, Mandi is designed to handle mainly the allocation of resources to users applications, while the job submission, monitoring and execution are left to user brokers. Mandi's services are available through platform independent APIs implemented by using Web Services.

(c) **Fault tolerance:** Mandi can handle failures at two stages: during trading, and during reservation. The failure during reservation can occur due to network problems, and over subscription of resources. In the case of network problem, the failed resource requests will be considered in the next scheduling cycle. The reservation failure due to resource oversubscription is handled by consumers and providers.

To avoid failures during trading, the resources are not reserved, unless all the tasks of an application are mapped. To avoid allocation of one resource to multiple applications, one compute resource is allowed to be traded only in one negotiation.

In addition, the persistence database protects the Market Exchange against failure during trading. The state of Mandi is periodically saved in the database. Thus, Mandi can resume its work from where it was left.

(d) **Scalability:** Most of the Mandi's components work independently and interact through the database. This facilitates the scalable implementation of Mandi as each component can be distributed across different servers accessing a shared database. Since, Mandi handles only the resource allocation and delegates the management of job submission and execution to the participating brokers and providers RMS, thus most of threads in Mandi are light weight and short lived.

8.3.2 Architectural Components

The architecture proposed in this work is inspired by the concepts of the 'Open Market' where any user can join, sell and buy their compute services. Figure 8.1 shows the service oriented architecture design of the Mandi's framework and its main services. Mandi is organised into two main services i.e. the user services, and the core services consisting of the meta-broker service the reservation service and the database service. The core functionality of each layer is described in the following sections. Each of the services can run on different machines independently, and communicate with only database service.

User Services

The user services hide all the internal components of Mandi and implement all the services visible to market participants. The services of Mandi are exposed to consumers and

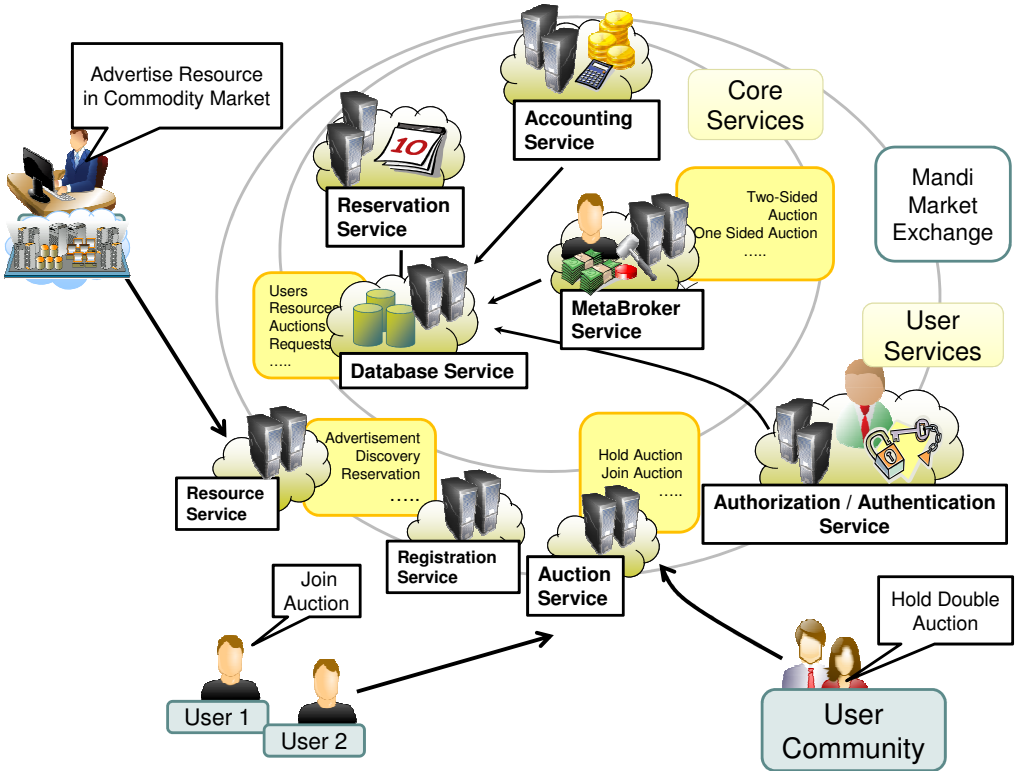


Figure 8.1: Mandi Architecture

resource providers through platform independent APIs which are implemented using Web Services. The following market services are provided to users:

(a) **Registration Service:** Users need to register before they can access the exchange's services. The users details are maintained in the storage layer, and are needed for authentication and authorisation.

(b) **Auction Service:** This service allows a user to join any auction and bid for the items. Hold Auction service allows users to specify the auction types which they are allowed to initiate. Mandi can conduct two classes of auctions, i.e., one-sided auction and two-sided auctions. In the case of two-sided auctions, multiple consumers and providers can choose to participate and get matched.

(c) **Resource Services:** Service Discovery and Service Reservation allow consumers to find services of their requirements and reserve them. This feature is added to integrate commodity market model within Mandi. Advertisement Service allows resource providers to advertise their cloud resources (number of CPUs and time at which they will be available).

(d) **Authentication and Authorization Service:** This service allows users to login into the market exchange and authorises them to access other Mandi services.

Core Services

These services consist of the internal services of the market exchange. Its core components are the meta-broker, the accounting and the advance reservation system.

(a) **Meta-Broker Service:** The initiation of any auction is managed by the meta-broker service. It conducts the auction and announces the auction winner through Advance Reservation service. The auction can either be one sided or two sided. Thus, two components i.e. Two-Sided Auction and One-Sided Auction are provided to add customised auction protocols within Mandi.

(b) **Advance Reservation Service:** It informs the resource providers about the match, reserves the advertised (matched) service, and gets the reservation id (referred as Ticket in Chapter 3) that is used by consumer to submit his application.

(c) **Accounting Service:** It records the trading information of each user. It also records the information about the failed and successful transactions.

(d) **Database Service:** This service is the interface between the persistence database and other agents such as the web interface, the advance reservation and the meta-broker. Its main objective is to maintain all the trading information such as transaction history, users details, auctions, compute resources for leasing, and user requests. It enables the recovery of Mandi in the case of unexpected failure, and is also useful in synchronising exchange's various components.

8.3.3 High Level Description

Mandi can initiates various negotiation protocols to match multiple user requests to the provider's ask. The negotiation protocols can be requested by users participating in the Mandi. For example, Figure 8.1 shows any user community can request *Auction Service* to hold a Double auction. In this case, the providers advertise their resources with their price (*aka* asks). Consumers submit their bids to show their interest in leasing the advertised resources. All the bids and asks are stored in the database which will be accessed at the end of auction for calculating the winner bids.

The *Meta-Broker*, which is the main agent of Mandi, coordinates the matching of asks and bids, and trading between auction participants. At the end of auction, the meta-broker finds out the winners and sends the reservation requests to the *Reservation Service* of Mandi. The *Reservation Service* informs the resource providers and consumers about the auction result. The information about reservations is stored within Mandi using *Accounting Service*.

8.3.4 User Interaction Phases

To understand the interrelationships between Mandi's services, it is necessary to see how they interact in different use-cases. There are several important use-cases when interacting with Mandi. There are two types of users trading in Mandi: a) consumers who need compute resources to execute their applications, and b) resource providers who are providing their infrastructure as service.

(a) **Registration:** Every user needs to register and submit their personal details such as name, organisation address, credit card details and contact phone number. After registration users are issued a login id and password to access exchanges services. The user details are stored in the persistence database and needed mostly for the accounting and logging purposes. These details can be accessed by other entities through User Catalogue component of the *Database Service*.

(b) **Resource Advertisement:** Any user can advertise the compute resource available for leasing. Each resource is assigned a unique identifier that is registered in Service Catalogue for discovery purposes. The user is required to give information such as how many CPUs/Virtual Machines(VMs) are available for processing and what are their configurations. The user also needs to inform Mandi using what trading protocol he/she wants to negotiate with resource providers. If the user selects the commodity market model, then the leasing price of the resource should be given while advertising the resource. If the user selects the auction model, then the auction type is needed to be selected by the user. All the information is stored in the storage layer that is accessed by other components of Mandi using *Database Service* for allocation purposes.

(c) **Service Discovery:** Users can discover resources that are advertised in Mandi through the *Discovery* component of *Resource Service*. To find the resource of their choice, users just need to give the configuration of compute resource they are interested in and when they want to lease. Mandi will search the Resource Catalogue service for the required resources, and send the matched resources with their trading protocol information to users. Users can select the resources of their choice and can utilise either *Resource Service* or *Auction Service* for leasing the resource.

(d) **Leasing Resources in Commodity Market:** To allow the integration of commodity market model in the Mandi market exchange, the reservation service is added to allow users to directly lease the available resources. A user provides the identifier of the resource to Mandi that he/she is interested in leasing. The users lease request is added in the Lease Request Catalogue which is regularly accessed by the Advance Reservation service. The reservation service does the final allocation of resources

by informing the resource provider and adding the information in the persistence database for accounting purposes. After allocation the leased resource and request are removed from the corresponding catalogues.

(e) **Conducting an Auction:** To hold an auction, first a user needs to get the type of auctions currently supported by the market exchange. Then, the user can send a request to hold the particular type of auction with details such as auction item, minimum bid, and auction end time. If the auction item is a compute resource, the user is required to specify the time for which CPU/VM will be leased. All the auction requests are stored in the database. Depending on the chosen auction protocol, an auction thread (with a unique identifier) is initiated by the Meta-Broker service. After initiation of the auction thread, the unique identifier (AuctionId) is sent back to the user (auction holder). The auction thread waits till the auction end time is reached. Users who want to bid in the auction need to provide the AuctionId that can be discovered using Join Auction service. Depending on the auction rules, the user is also allowed to resubmit updated bid. At the end of the auction, the auction thread collects the bids and executes the winner selection algorithm and sends the reservation request to the *Reservation Service*. The *Reservation Service* creates a contract for accounting purposes, and informs the participant about the auction outcome.

8.3.5 Implementation Details

The Class design diagram which illustrates the relationship between Mandi's Objects is depicted in Figure 8.2. The objects in the Mandi can be broadly classified into two categories - entity and workers. This terminology is derived from standard business modelling processes [51]. Entities exist as information containers representing the properties, functions and instantaneous states of the various architectural elements. The entities are stored in the database and are updated periodically. Workers represent the internal functionality of Mandi, that is, they implement the actual logic and manipulate the entities in order to allocate resources to applications. Therefore, workers can be considered as active objects and the entities as passive objects. The following sections take a closer look at some of the important entities and workers within Mandi.

Entities

(a) **User:** The User class stores information about the participant members (consumers and providers) of Mandi. This information is used for authentication and authorisation. From the point of view of the exchange, any user can act as consumer or provider thus there is no special field to differentiate between them in Mandi.

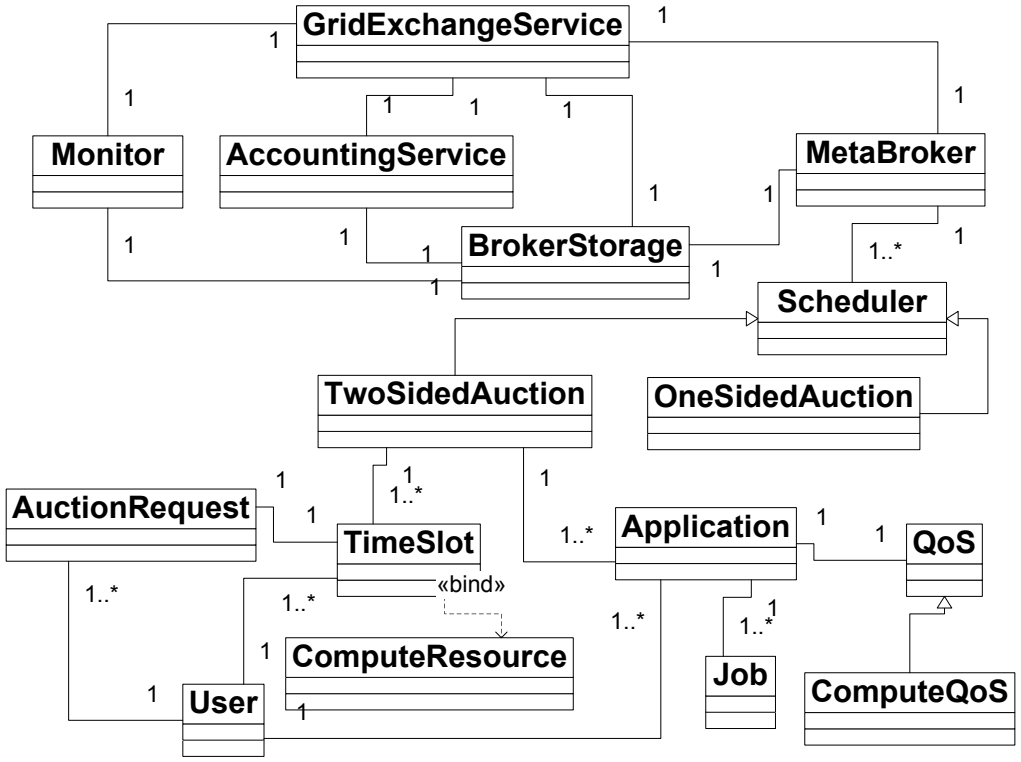


Figure 8.2: Mandi Class Design Diagram

Each user can advertise multiple compute resources, and submit multiple lease and auction requests.

(b) **TimeSlot and ComputeResource:** The TimeSlot Class is used to represent of compute resources available for leasing. The "TimeSlot" indicates how much time and how many CPUs are available. Each "TimeSlot" is associated with one compute resource that is a representation of a set of CPUs or virtual machines advertised by the resource provider. If a resource provider has conducted an auction for inviting bids for the time-slot, then the AuctionType and the AuctionID attributes will be used to store the auctions information. Each "TimeSlot" can be associated with only one auction.

(c) **Auction Request:** All the information for holding an auction for any commodity advertised by a user is represented using the AuctionRequest Class. Every auction is identified by a unique identifier i.e. auctionID. In economics, generally bids in auctions are considered in the form of monetary value. But in the case of computing service, a bid can be a more generalised form depending on the requirements of an auction holder. For example, a user holds an auction to find a resource provider who can lease the compute resource with minimum delay and within the specified budget. Thus, the user can invite bids in terms of the start time of the resource

lease. Mandi provides facilities to define different types of auctions which can be implemented by extending the TwoSidedAuction and OneSidedAuction classes. Each auction mechanism is specified by its winner selection algorithm.

To enable co-existence of multiple auction based negotiation with different matching and pricing strategies, the AuctionRequest class contains the "auctionType" attribute informing Mandi which auction user wants to hold.

(d) **Application:** The Application class abstracts the resource requests of the user's application that consists of the total number of CPUs required, QoS requirements, deadline, and budget. The "deadline" attribute represents the urgency of the user to get his/her application finished. The "QoS" is an abstract class that can be extended to codify special application requirements such as bandwidth. Each application can consist of several jobs that may differ in their resource requirements such as execution time. To allow users to submit different application model requirements such as parameter sweep and parallel application, in Mandi, each application is associated with the "appType" attribute that will be considered while matching an application with a resource. The application object also stores the information about the auction in which the user (consumer) has opted to participate for leasing resource for its application. Each application is allowed to participate in only one auction.

Workers

(a) **MetaBroker:** MetaBroker is the first component in Mandi to be started that instantiates other worker components, and manages their life cycles such as Scheduler and Monitor. The BrokerStorage is the front end to the persistence system and implements interfaces used to interact with the database. Another function of the MetaBroker is to periodically get the list of current auction requests from the database and start a scheduling thread for clearing each auction.

(b) **GridExchange Service:** The GridExchange Service is a Web Service interface that enables users to access various services of Mandi. The services that are available to users are registration, submission of application and time slots, holding and joining auctions, and discovering services and getting service reservations. The GridExchange Service interacts with the BrokerStorage class to access the persistence database. The example sequence of operations for user registration is shown in Figure 8.3. The UserBroker sends a registration request to the exchange using the GridExchange web service. It submits the preferred login name and password. The GridExchange service gets the registered user list from the database and checks whether the user is registered or not. If the user is not registered, it sends a reply back to user broker with "registration success" message.

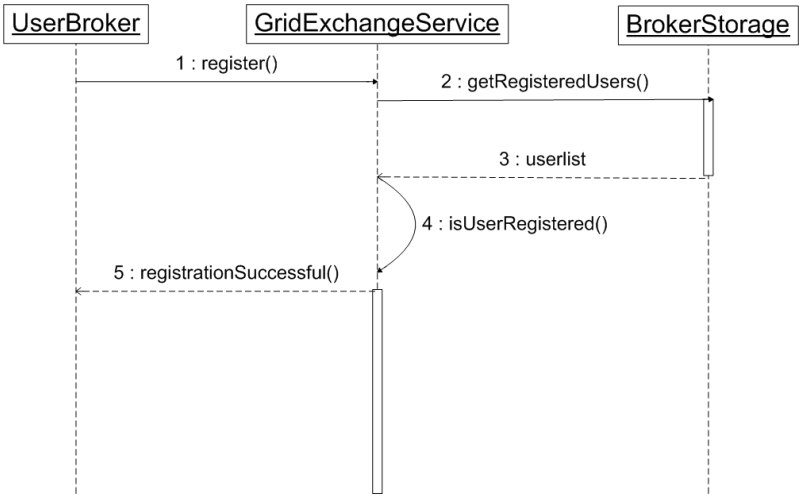


Figure 8.3: Registration Process

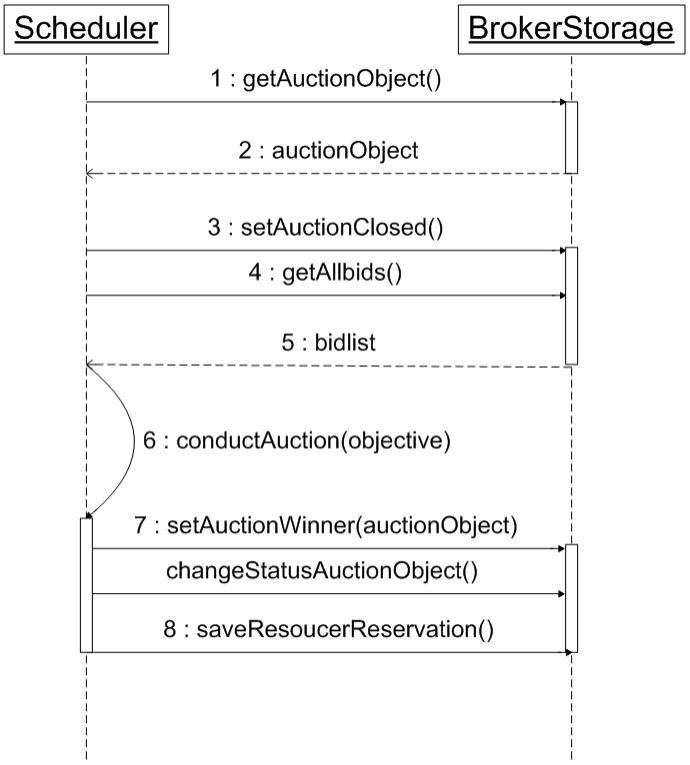


Figure 8.4: Scheduling Sequence

(c) **Scheduler:** For each market model, the Scheduler matches the user application to the advertised compute resources and also decides the price for executing the application. Figure 8.4 shows the basic steps that are performed by the Scheduler. The Scheduler gets the auction object (can be in the form of a timeslot or an application) from the persistent database and the list of all the bids submitted for the auction. The Scheduler sets the auction status to "closed" to prevent any further bid submission to the auction. Depending on the auction type and objective, the

winning bid is chosen and the trading price is calculated. The status of winning bid is changed to "matched" from "unmatched". The match is saved to database in the form a reservation request which will be used by the Monitor to inform/reserve resources on the compute resource. The function of the Monitor is described in detail below.

(d) **Monitor (*aka* advance reservation):** The Monitor keeps track of all the reservation request in the database, as shown in Figure 8.5. The Monitor periodically requests all the reservation requests from the persistent database. It uses Web Services to send SOAP messages to the resource provider, which informs the provider of the matching of the user application to the advertised timeslot (compute service). In the return, the Monitor gets the reservationID from the provider. The reservationID is used by the consumer to access the compute services offered by the resource provider. It represents the time-slot reserved and is also the security key for accessing the resource. After getting the reservationID, the Monitor will set all the reservation details in the user application object stored in the persistent database. The consumers (using brokers) can access the reservation information by using the GridExchange Service.

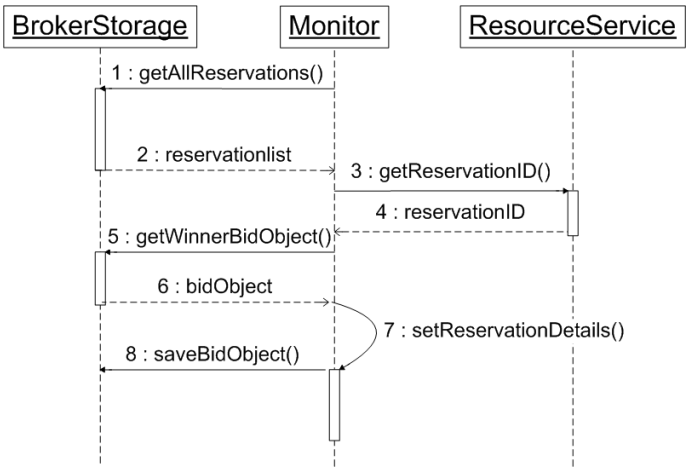


Figure 8.5: Reservation Process

8.4 Prototype and Performance Evaluation

In order to evaluate the performance of Mandi and provide a proof of concept of its architecture, we implemented a prototype and tested it by using Aneka [35] as a service provider. In this section, we will give an overview of the components composing the system used for testing and discuss the performance evaluation.

8.4.1 System Details

Mandi

Mandi has been implemented in Java in order to be portable over different platforms such as the Windows and Unix operative systems. From an implementation point of view Mandi is composed of a collection of services that interact by means of a persistence layer represented by the HSQL database. The system is accessible from external components through a Web Service that has been deployed by using Apache Axis2 on a TOMCAT web server (v.5.5). The Web Service interface makes the interaction with Mandi platform independent. The current prototype support three type of trading mechanisms: i) *First Bid Sealed Auction*; ii) *Double Auction*, and iii) *Commodity market*.

Aneka

On the provider side, Aneka [35] has been used and extended to support the reservation and advertisement of slots on Mandi. Aneka is a service-oriented middleware for building Enterprise Clouds. The core component of an Aneka Cloud is the Aneka container that represents the runtime environment of distributed applications on Aneka. The container hosts a collection of services through which all the tasks are performed: scheduling and execution of jobs, security, accounting, and reservation. In order to support the requirements of Mandi a specific and lightweight implementation of the reservation infrastructure has been integrated into the system. This infrastructure is composed by a central reservation service that provides global view of the allocation map of the Cloud and manages the reservation of execution slots, and a collection of allocation services on each node hosting execution services that are in charge of keeping track of the local allocation map and of ensuring the exclusive execution for reserved slots. The reservation service is accessible to external applications by means of a specific Web Service that exposes the common operations for obtaining the advertised execution slots and reserving them.

Client Components

The client components are constituted by a simple Web Service client that generates all the resource requests to Mandi.

8.4.2 Performance Evaluation

We evaluated the performance of Mandi in terms of overhead caused to the system due to the interaction between the internal components and Mandi's interaction with users requests and provider's middleware. As discussed previously, Mandi is designed to handle multiple market models concurrently and exposes a service oriented interface to handle

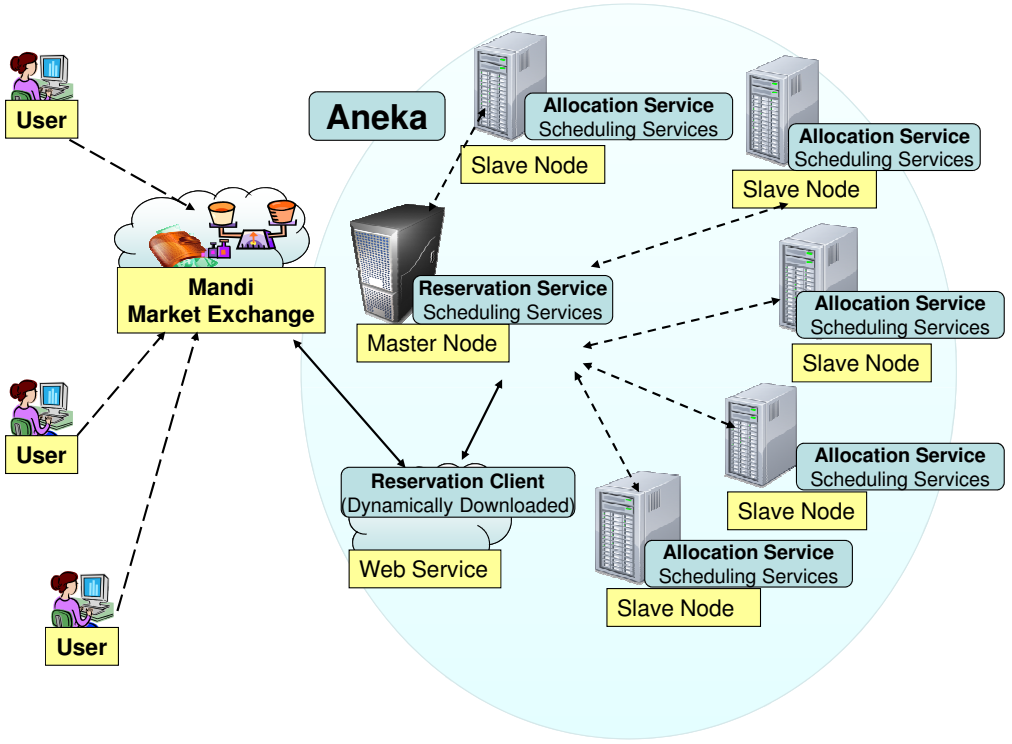


Figure 8.6: The Topology of Testbed

users requests and reservations of resources. Thus, to evaluate the scalability of Mandi, the first set of experiments examines the CPU and memory requirements of our implementation of Mandi. However, the performance of Mandi is also determined by how quickly and how many simultaneous user requests can be handled. Hence, the second set of experiments evaluates time incurred in resource request submission (which is initiated from client machine) and resource reservation (which involve negotiation of Mandi with providers).

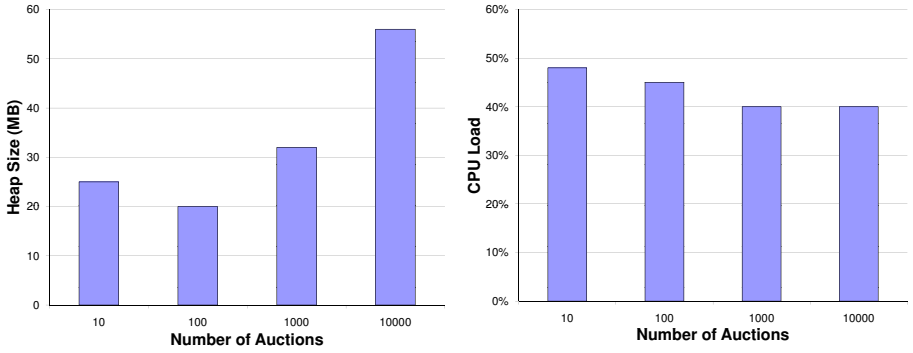
The experimental setup for this evaluation is characterised as follows:

- An instance of Mandi has been deployed on 2.4 GHZ Intel Core Duo CPU and 2 GB of main memory running the Windows operative system and Java 1.5. The HSQL Database was configured to run on the same machine. The performance of Mandi evaluated using JProfiler [144] profiling tool.

- The Aneka setup was characterised by one master node and five slave nodes. The reservation infrastructure was configured as follows: the master node hosted the reservation service while each of the slave nodes contained an instance of the allocation service. Each container has been deployed on a DELL OPTIPLEX GX270 Intel Core 2 CPU 6600 @2.40GHz, with 2 GB of RAM and Microsoft Windows XP Professional Version 2002 (Service Pack 3). As a result the reservation infras-

tructure can support ten concurrent execution lines (one per core). The topology of resources is given in Figure 8.6.

Memory Usage and CPU Load



(a) Memory Required by Mandi (b) CPU Usage by Mandi

Figure 8.7: Performance of Mandi for 50,000 clearance requests

The main threads running in Mandi are: i) MetaBroker, which initiates other threads and controls the overall execution of Mandi, ii) Monitoring Thread, and iii) Scheduler Threads, which dynamically vary based on the number of auctions. Thus, the performance of Mandi is highly dependent on the number of auctions conducted concurrently. Thus, to evaluate the performance of Mandi, we varied the number of auctions from 10 to 10,000 that are conducted over period of 5 seconds. For this experiment, we generated 50,000 resource requests for matching. Each resource request is mapped to an auction using uniform distribution. Figure 8.7 shows the graphs of the memory and CPU usage by the broker over a period of 5 Second run. In Figure 8.7(b), the variation in CPU usage is about 10% with increase in number of auctions. This is because scheduler threads conducting auctions are short lived and has comparable CPU needs. The little higher value of CPU usage in the case when 10 auctions are conducted is due to the large number of resource request per auction (50,000/10) needed to be matched.

In figure 8.7(a), we can see how memory usage of Mandi is increasing with the number of auctions. For instance, the memory usage increases from 32 MB to 56 MB when the number of auctions increases from 1000 to 10000. Therefore, there is only a 2 times increase in memory usage for 10 times increase in the number of auctions. This is due to the fact that the auction thread loads resource requests from database only when a decision for the auction winner needs to be taken. In addition, the memory is freed for all resource requests participating in the auction as soon as auction finished executing. This reduces the memory occupied by resource request objects waiting to be matched.

Overhead in Interaction with Resource Provider and Consumer

Two experiments were performed; one for measuring the resource request submission time, and the other for reservation time and free slot advertisement by the provider middleware. All interactions between different entities i.e Mandi, consumer, and provider middleware is using web service. To measure these parameters, we used JMeter tool that generate SOAP messages to test the performance of web services. We generated SOAP messages until no more connection can be initiated with the web service located at Mandi and resource provider's site. In case of interaction with Mandi's web service, about 750 concurrent resource submission requests were generated, while in case of interaction with Aneka reservation web service about 100 concurrent requests were generated. Table 8.1 shows the time taken to serve a request by web service in milliseconds. Overhead in terms of time for resource request submission is only 11.75 ms. The time taken by Aneka web service to serve free resource and reservation requests is much longer because each reservation request can trigger the interaction between the reservation service on the master node and the allocation service on the slave node where the reservation is allocated. This interaction implies the communication between two different containers and varies sensibly according to the network topology.

8.4.3 Discussion

The performance results indicate the scalability of current prototype of Mandi which is able to clear about 50,000 resource requests and 10,000 auctions in about 5 seconds. The major bottleneck in the scalability of Mandi's architecture is the shared database. The database constraints the number of multiple and concurrent accesses which is also the reason that experiments over 50,000 resource requests are not conducted. In addition to that the database can be cause of single point failure of whole system. The distributed databases which use replication and load balancing techniques can be helpful in increasing the scalability of the system.

Table 8.1: Overhead due to Interactions of Mandi

Web Service Request	**Service Time/request (ms)**
Resource Request Submission	11.75
Getting Free Resources	30
Resource Reservation	240

8.5 Summary

The presence of IT demand and supply in utility oriented Clouds and Grids led to the need of a market exchange that can ease the trading process by providing the required infrastructure for interaction. In this chapter, we introduce a novel market exchange framework named ”Mandi” for facilitating such trading. We identify the essential technical and market requirements, and challenges in designing such an exchange. The architecture and implementation of Mandi is comprehensively described and evaluated. The two experiments performed measure the effect of design choices on the performance of Mandi, and overhead time incurred in the interaction between the consumer and provider through Mandi. The experiments show that Mandi can scale well and can handle many concurrent trading models and resource requests. We can thus conclude that the overhead generated for matching a large number of resource requests in concurrent auctions is minimal. The only limit to the scalability of the system is the persistence layer, which also constitutes the single point of failure. In order to address this issue, a more efficient database server and a solid replication infrastructure has to be put in place.

Chapter 9

Conclusions and Future Directions

This chapter summarises our objectives and work done in this thesis. Our main findings and lessons learned are discussed along with their significance. In this chapter, this thesis concludes with a discussion on the future work that emerged during this research but are loosely connected and outside its scope.

9.1 Summary

This thesis began with the introduction of utility Grids which are becoming the mainstream infrastructure for executing large scale applications from academia and industrial enterprises. Many challenges are pointed out in solving the problem of resource management and scheduling in utility Grids. The literature review identified a lack of research on the coordinated scheduling of multiple applications which have conflicting QoS requirements, across multiple resource sites. Thus, we set out to investigate how to design market-oriented meta-scheduling algorithms and mechanisms which can reconcile and coordinate users' QoS requirements while maximising the utility for both users and resource providers. To solve the problem of market-oriented meta-scheduling, the following objectives are defined:

- To investigate an architectural model for a market-oriented meta-scheduler to coordinate resource demand,
- To design the meta-scheduling algorithms and mechanisms that can reconcile the resource demands from users with conflicting requirements,
- To investigate how other system-centric metrics such as response time will impact both the participants' utility.

The first objective is achieved in Chapter 3, which proposed the meta-broker architecture and evaluated its advantages against a completely decentralised model, where each

personalised broker competes to access resources in a utility Grid. We showed in that chapter how the meta-broker, not only maximises the successful execution of user applications, but also maximises the resource utilisation.

This thesis accomplished the second and third objectives by studying the problem in three scenarios: i) to maximise users' utility, ii) to maximise providers' utility, and iii) to maximise both users' and providers' utility. Chapter 4 and 5 discussed the problem from user's point of view and proposed several heuristics to maximise the user's utility in terms of performance and monetary costs. Experimental results showed how the trade-off between two conflicting goals can be managed by aggregating them within a trade-off metric.

Chapter 6 examined mechanisms which can maximise a provider's utility by minimising the maintenance costs in terms of energy consumption of CPUs. This chapter also presented mechanisms that minimise carbon emissions from the environment's perspective. Chapter 7 addressed the problem of meta-scheduling to maximise both users' and resource providers' utility using analytical and experimental approaches. In the next section, we summarise the main contributions of this thesis with lessons learned and their significance.

9.2 Lessons Learned and Significance

In the beginning of this thesis, a taxonomy was developed to classify the common market-oriented scheduling mechanisms based on allocation decisions, mechanisms' objective, market model, application model, and participant focus. The taxonomy provided the basis for comparing market-oriented scheduling systems and technologies. This comprehensive classification not only enhances the understanding of recent developments in utility Grids, but also provides an insight into research gaps which still need to be addressed.

Based on the literature study, we pointed out that a lack of coordination between personalised brokers can lead to the contention for cheap and efficient resources. Our simulation study in Chapter 3 showed that the contention can even result in underutilisation of resources. Thus, this thesis proposed a market-oriented meta-scheduler architecture called "Meta-Broker". The meta-broker architecture is semi-decentralised, thus only scheduling decisions are made by the meta-broker, while job submission and monitoring is performed by personalised user brokers. This hybrid design makes the meta-broker system light weight, and thus, it can handle a large number of scheduling requests, while allowing local autonomy to its participants. The meta-broker allows the inclusion of various strategies for resource selection and allocation. From our simulation study, we learned that the meta-broker architecture can maximise the utilisation of resources by decreasing the deadline violation of applications.

Motivated by these preliminary results, this thesis investigated the meta-scheduling algorithms that can minimise the combined spending of users with QoS requirements such as deadline and budget. To analyse challenges involved, the meta-scheduling problem of scheduling multiple applications on multiple resources is modelled using Linear/Integer Programming model. Based on this model, a novel genetic algorithm, LPGA, was presented to minimise the combined spending of users in executing their applications. This thesis also reveals how approximation algorithms such as MMC in Grid environments can be designed to achieve cost optimisation.

In Chapter 5, we relaxed the deadline constraint, and studied the problem in a scenario where users also wanted to optimise system metrics such as makespan while minimising monetary cost. The challenge was to schedule various applications in a coordinated manner by satisfying as many users as possible with the goal of minimising the monetary cost and time of using resources for all Grid users. This scheduling problem, which aims to minimise the cost and time of using resources for all concurrent users, is found to be NP-hard due to its combinatorial nature. The problem becomes more challenging when a user has to relax her QoS requirements, such as makespan, under limited budget constraints. Users may prefer cheaper services with a relaxed QoS, if it is sufficient to meet their requirements. Thus, a user has to choose between multiple conflicting optimisation objectives [102]. This problem is not only strongly NP-hard, but also non-approximable [101], i.e., it cannot be approximated in polynomial time within a reasonably good precision. Moreover, the scheduling in utility Grids needs to be online, which further adds to the challenge. Thus, this thesis proposed three meta-scheduling online heuristics, such as MinMin Cost Time Trade-off (MinCTT), Sufferage Cost Time Trade-off (SuffCTT), and Max-Min Cost Time Trade-off (MaxCTT), to manage the trade-off between overall execution time and cost, and minimise them simultaneously on the basis of a trade-off factor. The trade-off factor is used to resolve the conflicting goals of a user. It indicates the priority of optimising cost over time. The proposed heuristics were evaluated by an extensive simulation study, which analysed the best heuristic to adopt according to different user preferences. The heuristic MinCTT gave the lowest makespan and cost in almost all scenarios considered.

The algorithms proposed in Chapter 4 and 5 can be used in scheduling applications of users from the same community (scientific or industry) on Cloud resources provided commercially by companies such as Amazon. The Cloud resources are virtualised computing infrastructure which can be leased on the hourly basis. The organizations such as Morgan Stanley, having more than one research divisions, can directly employ our approaches to schedule their applications from different divisions in a coordinated and cost-effective manner.

In Chapter 6, we investigated the market-oriented scheduling problem in the resource

provider's context. This thesis gave a novel approach to maximise the resource provider's profit (utility) by intelligent energy-efficient meta-scheduling of applications across heterogeneous resource sites. Thus, the proposed approach reduces the maintenance cost which results in higher profit rather than selecting profitable users. This scenario is very timely since high maintenance costs in data centers due to the increasing price of electricity have become a major concern. To maximise profit for the provider, this thesis proposed three simple, yet effective generic scheduling policies that can be extended to any application, processor, and server models so that they can be readily deployed in existing data centers with minimum changes. Our generic scheduling policies can also easily complement any of the existing application-specific, processor-specific, and/or server-specific energy-saving policies that are already in place within existing data centers or servers. We identified some of the factors such as electricity rates, energy efficiency of data centers and carbon emissions which play an important role in minimising the maintenance cost for resource providers. Thus, the thesis addressed an important topic of balancing profit and carbon emissions for globally distributed utility computing environments such as Clouds. This thesis highlighted the need to consider the heterogeneity in data centers when mapping workloads, and to conserve energy when scheduling and running the workloads.

In Chapter 7, this thesis investigated the meta-scheduling problem of simultaneously maximising both users' and resource providers' utility. A question to answer was "Can we design a meta-scheduling approach which can benefit users by satisfying their QoS and benefit resource providers by increasing their utilisation?". Another more specific question to answer was "How the consideration of other system-centric metrics such as response time will impact the participants' utility". This thesis proposed a meta-scheduling mechanism (Double Auction-Inspired Meta-Scheduling Mechanism) for parallel applications that takes advantage of both auctions and system-based schedulers to simultaneously maximise satisfied users, and resource providers by evenly distributing load across all the resources. To achieve this, valuation metrics were designed using system parameters that commodify the available resource share and the user's application requirements so that they can be compared and matched using principles similar to Double auctions. To investigate the impact on system metrics, this meta-scheduling problem was analysed through a queuing theory based analytical model. This thesis demonstrated that by considering both system metrics and market parameters we can enable more effective scheduling which will benefit both users and resource providers. This thesis also demonstrated how classical economic models, when adapted suitably, are able to deal with multiple QoS requirements of the users more effectively than state-of-the art algorithms used in existing schedulers. This motivates further exploration of other economic models to solve particular problems in job scheduling.

Finally, we realised the proposed architecture by presenting a system prototype as part of Mandi Grid exchange . We implemented it using Web Services to make this platform independent. A web interface is also provided so that users can check their account details and advertise resources. Mandi gives flexibility to its users not only in terms of trading protocol but also allows co-existence of multiple negotiation between users and providers.

In summary, the planning and scheduling algorithms proposed in this thesis have taken into account both costs and capabilities of Grid resources while meeting users QoS requirements. Hence, this thesis successfully demonstrated the feasibility of using market-oriented meta-scheduling for multiple applications in utility Grids, and made significant contributions towards the advancement of the discipline.

9.3 Future Directions

In this thesis, the problem of market-oriented meta-scheduling of multiple applications with QoS constraints on heterogeneous and distributed resources was addressed. However, there are still open issues that can serve as a starting point for future research.

9.3.1 Resources with Different Pricing Models

In this thesis, we considered a fixed pricing model while scheduling multiple applications. In recent years, many research studies [148] [154] has been done on pricing the resources in utility Grids. Each provider can employ different dynamic pricing policies for maximising their profit. For example, Amazon [6] uses two types of pricing models; a) spot pricing and b) fixed pricing. Each of these models gives some advantages and disadvantages to end users. For example, the spot pricing can be exploited by meta-broker to maximise the user's profit but it reduces the chances of task being executed successfully. In such environments, not only the current but also future status of resources needs to be considered to reduce the user's spending. Hence, there is a need to understand the effect of using different pricing models on cost-based scheduling, and design novel scheduling policies to handle such heterogeneity. The time factor will again play an important role in this study.

9.3.2 Scheduling with Pre-emption

Throughout this thesis, we considered a non-preemptive execution model to avoid delays caused in the transfer of applications from one resource provider to another. However, there are some cases where checkpoints can be used to restart the application from the point, where execution was stopped. This has become more feasible due to recent technological advancement in Virtual Machines [31]. In such cases, the consideration of migra-

tion might be useful to enhance the utility of users and resource providers. The provider can preempt those applications which are less profitable and accept more profitable ones. Similarly, the meta-broker can migrate the application from expensive resources (fast) to cheaper ones (slow). The migration of such jobs would be a challenging task since we also need to consider other issues such as the application model, network delays, resource status and checkpointing.

9.3.3 Network and Data-Aware Application Meta-scheduling

In scientific environments such as High Energy Physics (HEP), there are several applications that require petabytes of data from various repositories distributed across various nations. The meta-scheduling of these applications competing for compute and storage resources can be very challenging due to the highly dynamic nature of network. In addition, computation should be ideally located near to storage, thus decreasing the delays in the execution. If the scheduling decisions are made just on the basis of either data size or computation time, the resultant schedule can lead to resource wastage in terms of network bandwidth, and performance degradation due to large execution delays. Moreover users need to pay for both data transfer and computation. These challenges soar up when there is contention for resources by concurrent users with similar applications. Thus, intelligent meta-scheduling approaches that take into account not only monetary execution costs, but also reconciling the competing storage, network and computation demand of users are required.

9.3.4 SLA based Meta-Scheduling in Cloud Computing Environments

Cloud computing has emerged as an important utility computing paradigm that provides highly flexible on-demand access to commercial resources. Being a commercialised environment, the formal SLA acts as the basis for resource sharing. In the case when users require resources from multiple resource providers, multiple levels of SLAs may exist. From the user's perspective, it is challenging to handle any SLA violation because it can recursively affect other SLAs. From the provider's perspective, the challenge is to manage SLAs in such a way to ensure profit maximisation. Thus, admission control plays an important role.

9.3.5 Energy-efficient Meta-Scheduling

Energy efficiency is important for the development of utility computing and environmental sustainability. In Chapter 6, we showed how meta-scheduling can also help in reducing energy consumption to a great degree. This finding has given a new research perspective

into scheduling across data centers. For further efficiency within a data center, different techniques for energy consumption, such as turning servers on and off, need to work simultaneously. This requires more technical analysis of the effect of various strategies on the execution delays and power consumption for servers, as well as the effect on the reliability of computing devices. The benefit of virtualised environments can also be harnessed. Virtualisation makes it easier to consolidate many applications on fewer physical servers. In addition, in future research, it is important to consider the energy (and potential latency) overhead of moving data sets between the data centers, in particular for data-intensive HPC applications. This overhead can be quite significant depending on the size of the data set and the activity of the workloads.

9.3.6 Scalable Meta-Broker Architecture

The meta-broker coordinates communication between multiple resource providers and user brokers. Being a common entry point for accessing global information about resources, the failure of the meta-broker can result in the failure of the entire system. In addition, in the case of high load, the computation required to make scheduling decisions can cause long execution delays. Thus, the scalability of the meta-broker can become a bottleneck in the system's performance.

To resolve the scalability issue, investigation is required on the most suitable architecture for the meta-broker. A decentralised architecture such as P2P can be incorporated so that the single point of failure can be avoided and fast scheduling decisions can be made by sharing information between peers. In addition, the communication protocol between peers needs to be investigated.

9.3.7 Mixed Application Model with QoS Requirements

It will be interesting to enhance all heuristics to schedule jobs with different application models such as workflows. Different application models forces different QoS needs on the scheduling strategy. For example, workflow and bag-of-task application, which consist of many independent tasks, may not require re-execution of the entire application in case of failure. On the other hand, in the case of parallel application, any failure will require the re-execution of the entire application. Thus, it is important to study the challenge of scheduling applications by reconciling the QoS needs based on their model. In addition, the proposed heuristics in this thesis can also be enhanced by considering more QoS needs such as memory and network bandwidth.

REFERENCES

[1] D. Abramson, R. Buyya, and J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computing System*, 18(8):1061–1074, 2002.

[2] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen. Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. *Tamkang Journal of Science and Engineering*, 3(3):195–208, 2000.

[3] D. Allenotor and R. Thulasiram. Grid Resources Pricing: A Novel Financial Option based Quality of Service-Profit Quasi-Static Equilibrium Model. In *Proceedings of 9th IEEE/ACM International Conference on Grid Computing*. IEEE Computer Society, 2008.

[4] J. Altmann, C. Courcoubetis, J. Darlington, and J. Cohen. GridEcon-The Economic-Enhanced Next-Generation Internet. In *Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France*, 2007.

[5] J. Altmann, M. Ion, A. Adel, and B. Mohammed. A Taxonomy of Grid Business Models. In *Proceedings of the 4th International Workshop on Grid Economics and Business Models, Rennes, France*, 2007.

[6] Amazon. Amazon Elastic Compute Cloud (EC2). `http://www.amazon.com/ec2/`, Aug. 2009.

[7] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. *Lecture Notes in Computer Science*, 2862:61–86, 2003.

[8] P. Andreetto, S. Andreozzi, G. Avellino, S. Beco, A. Cavallini, M. Cecchi, V. Ciaschini, A. Dorise, F. Giacomini, A. Gianelle, et al. The gLite Workload Management System. In *Journal of Physics: Conference Series*, volume 119, page 062007. Institute of Physics Publishing, 2008.

[9] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource Allocation in Federated Distributed Computing Infrastructures. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure, NV, USA*, 2004.

[10] C. Belady. In the Data Center, Power and Cooling Costs More Than the IT Equipment it Supports. *Electronics cooling*, 13(1):24, 2007.

[11] F. Berman and R. Wolski. The AppLeS Project: A Status Report. In *Proceedings of the 8th NEC Research Symposium*, Berlin, Germany, 1997.

[12] R. Bianchini and R. Rajamony. Power and Energy Management for Server Systems. *Computer*, 37(11):68–74, 2004.

[13] A. Bose, B. Wickman, and C. Wood. Mars: A Metascheduler for Distributed Resources in Campus Grids. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA*, 2004.

[14] D. Bradley, R. Harper, and S. Hunter. Workload-based Power Management for Parallel Computer Systems. *IBM Journal of Research and Development*, 47(5):703–718, 2003.

[15] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[16] J. Broberg, S. Venugopal, and R. Buyya. Market-oriented Grids and Utility Computing: The State-of-the-Art and Future Directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[17] M. Buco, R. Chang, L. Luan, C. Ward, J. Wolf, and P. Yu. Utility Computing SLA Management Based upon Business Objectives. *IBM Systems Journal*, 43(1):159–178, 2004.

[18] T. Burd and R. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proceedings of the 28th Hawaii International Conference on System Science, Maui, Hawaii, USA*, 1995.

[19] J. Burge, P. Ranganathan, and J. Wiener. Cost-Aware Scheduling for Heterogeneous Enterprise Machines (CASHEM). In *Proceedings of 2007 IEEE International Conference on Cluster Computing, Austin, Texas, USA*, pages 481–487, 2007.

[20] J. Butler, D. Morrice, and P. Mullarkey. A Multiple Attribute Utility Theory Approach to Ranking and Selection. *Management Science*, 47(6):800–816, 2001.

[21] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[22] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Proceedings of the IEEE*, 93(3):698–714, 2005.

[23] R. Buyya and K. Bubendorfer, editors. *Market-Oriented Grid and Utility Computing*. John Wiley & Sons Inc, New Jersey, USA., 2009.

[24] R. Buyya and M. Murshed. Gridsim: A Toolkit For the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[25] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm. *Software: Practice and Experience*, 35(5):491–512, 2005.

[26] R. Buyya and S. Vazhkudai. Compute Power Market: Towards a Market-Oriented Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, 2001.

[27] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[28] F. Cappello. Towards an International Computer Science Grid. In *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Paris, France*, 2007.

[29] C. Catlett. The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility. In *Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany*, 2002.

[30] S. Chapin, J. Karpovich, and A. Grimshaw. The Legion Resource Management System. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico, 1999.

[31] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle. Dynamic Virtual Clusters in a Grid Site Manager. In *Proceedings of the Twelfth International Symposium on High Performance Distributed Computing, Seattle, Washington, USA*, 2003.

[32] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Serve Resources in Hosting Centers. *SIGOPS Opering System Review*, 35(5):103–116, 2001.

[33] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):303–314, 2005.

[34] P. Chu and J. Beasley. A Genetic Algorithm for the Generalised Assignment Problem. *Computers and Operations Research*, 24(1):17–23, 1997.

[35] X. Chu, K. Nadiminti, C. Jin, S. Venugopal, and R. Buyya. Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications. In *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing, Bangalore, India*, 2007.

[36] B. Chun and D. Culler. Rexec: A Decentralized, Secure Remote Execution environment for Clusters. In *Proceedings of 4th Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, Toulouse, France*, 2000.

[37] B. Chun, C. Ng, J. Albrecht, D. Parkes, and A. Vahdat. Computational Resource Exchanges for Distributed Resource Allocation. Technical report, http://citeseer.ist.psu.edu/706369.html, 2004.

[38] A. Cobham. Priority Assignment in Waiting Line Problems. *Journal of the Operations Research Society of America*, 2(1):70–76, 1954.

[39] B. Cooper and H. Garcia-Molina. Bidding for Storage Space in a Peer-to-Peer Data Preservation System. In *Proceeding of 22nd International Conference on Distributed Computing Systems, Vienna, Austria*, 2002.

[40] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, et al. New Grid Scheduling and Rescheduling Methods in the GrADS Project. In *Proceedings of 18th International Parallel and Distributed Processing Symposium, New Mexico, USA*, 2004.

[41] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, USA*, 2001.

[42] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. *Lecture Notes in Computer Science*, 2537:153–183, 2002.

[43] E. David, R. Azoulay-Schwartz, and S. Kraus. Protocols and Strategies for Automated Multi-Attribute Auctions. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems , Bologna, Italy*, 2002.

[44] M. de Assunção and R. Buyya. Performance Analysis of Allocation Policies for InterGrid resource Provisioning. *Information and Software Technology*, 51(1):42–55, 2009.

[45] V. Di Martino and M. Mililotti. Scheduling in a Grid Computing Environment using Genetic Algorithms. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium, Fort Lauderdale, Florida, USA*, 2002.

[46] A. Dogan and F. Ozgiiner. Scheduling Independent Tasks with QoS Requirements in Grid Computing with Time-Varying Resource Prices. In *Proceedings of Third International Workshop of Grid Computing, Baltimore, MD, USA*, 2002.

[47] T. Dornemann, E. Juhnke, and B. Freisleben. On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China*, 2009.

[48] C. Dumitrescu and I. Foster. Gruber: A Grid Resource Usage SLA Broker. 2005.

[49] C. Dumitrescu, I. Raicu, and I. Foster. DI-GRUBER: A Distributed Approach to Grid Resource Brokering. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC'05)*, Seattle, WA, USA, 2005.

[50] A. Elyada, R. Ginosar, and U. Weiser. Low-Complexity Policies for Energy-Performance Tradeoff in Chip-Multi-Processors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(9):1243–1248, 2008.

[51] H. Eriksson and M. Penker. Business Modeling with UML: Business Patterns at Work, John Wiley&Sons, 2001.

[52] C. Ernemann, V. Hamscher, and R. Yahyapour. Economic Scheduling in Grid Computing. In *Proceedings of 7th International Workshop on Job Scheduling Strategies for Parallel Processing, Cambridge, MA, USA*, 2001.

[53] M. Etinski, J. Corbalan, J. Labarta, M. Valero, and A. Veidenbaum. Power-Aware Load Balancing of Large Scale MPI Applications. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing, Rome, Italy*, 2009.

[54] T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag, and L. Navarro. Decentralized Resource Allocation in Application Layer Networks. In *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan*, 2003.

[55] T. Eymann, M. Reinicke, F. Freitag, L. Navarro, Ó. Ardáiz, and P. Artigas. A Hayekian Self-Organization Approach to Service Allocation in Computing Systems. *Advanced Engineering Informatics*, 19(3):223–233, 2005.

[56] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-Sized Computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 13–23, New York, NY, USA, 2007.

[57] D. Feitelson. Parallel Workloads Archive. *URL http://www. cs. huji. ac. il/labs/parallel/workload*.

[58] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong. Theory and Practice in Parallel Job Scheduling. In *Proceedings of 1997 Job Scheduling Strategies for Parallel Processing, Geneva, Switzerland*, 1997.

[59] M. Feldman, K. Lai, and L. Zhang. A Price-Anticipating Resource Allocation Mechanism for Distributed Shared Clusters. In *Proceedings of the 6th ACM Conference on Electronic Commerce, Vancouver, Canada*, 2005.

[60] H. Feltl and G. Raidl. An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. In *Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus*, 2004.

[61] H. Feng, G. Song, Y. Zheng, and J. Xia. A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Dependent Tasks in Grid Computing. *Lecture Notes In Ccomputer Science*, pages 113–120, 2004.

[62] W. Feng and K. Cameron. The Green500 List: Encouraging Sustainable Supercomputing. *Computer*, 40(12):50–55, 2007.

[63] X. Feng, R. Ge, and K. W. Cameron. Power and Energy Profiling of Scientific Applications on Distributed Systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Los Alamitos, CA, USA, 2005.

[64] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.

[65] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[66] V. Freeh, D. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. Rountree, and M. Femal. Analyzing the Energy-Time Trade-Off in High Performance Computing Applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835, 2007.

[67] V. W. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer. Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, Los Alamitos, CA, USA, 2005.

[68] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, Jul 2002.

[69] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, New York, USA*, 2003.

[70] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal Power Allocation in Server Farms. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems, Seattle, WA, USA*, 2009.

[71] W. Gentzsch et al. Sun Grid Engine: Towards Creating a Compute Power Grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid, Brisbane, Australia*, 2001.

[72] K. Golconda, F. Ozguner, and A. Dogan. A Comparison of Static QoS-based Scheduling Heuristics for a Meta-Task with Multiple QoS Dimensions in Heterogeneous computing. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico*, 2004.

[73] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt. Best Practices for Data Centers: Results from Benchmarking 22 Data Centers. In *Proceedings of the 2006 ACEEE Summer Study on Energy Efficiency in Buildings*, Pacific Grove, USA, 2006.

[74] D. Grosu and A. Das. Auction-based Resource Allocation Protocols in Grids. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems, Cambridge, USA*, 2004.

[75] M. Harchol-Balter, M. Crovella, and M. C.D. On Choosing a Task Assignment Policy for a Distributed Server System. *Journal of Parallel and Distributed Computing*, 59(2):204–228, 1999.

[76] M. Harchol-Balter and A. Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems (TOCS)*, 15(3):253–285, 1997.

[77] G. Hardin. The Tragedy of the Commons. *Science*, 162(3859):1243–1248, 1968.

[78] D. Hausheer and B. Stiller. Peermart: The Technology for a Distributed Auction-based Market for Peer-to-Peer Services. In *Proceedings of the 2005 IEEE International Conference on Communications, Seoul, South Korea*, 2005.

[79] R. Henderson. Job Scheduling Under the Portable Batch System. In *Proceedings of the 1995 Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA*, 1995.

[80] J. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *Computer*, 33(7):28–35, 2000.

[81] E. Herness, R. High, and J. McGee. WebSphere Application Server: A Foundation for on Demand Computing. *IBM Systems Journal*, 43(2):213–237, 2004.

[82] W. Hoschek, F. J. Jaén-Martínez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 77–90, London, UK, 2000. Springer-Verlag.

[83] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *Proceedings of 1st International Workshop on Grid Computing, Bangalore, India*, 2000.

[84] C. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *Proceedings of the 2003 ACM SIGPLAN Conference on Programming Language Design and Implementation, Sweden*, 2003.

[85] E. Huedo, R. Montero, I. Llorente, D. Thain, M. Livny, R. van Nieuwpoort, J. Maassen, T. Kielmann, H. Bal, G. Kola, et al. The GridWay Framework for Adaptive Scheduling and Execution on Grids. *Software-Practice and Experience*, 6(8), 2005.

[86] O. Ibarra and C. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[87] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. Yocum. Sharing Networked Resources with Brokered Leases. In *Proceedings of the 2006 USENIX Technical Conference, Boston, MA, USA*, 2006.

[88] D. Irwin, L. Grit, and J. Chase. Balancing Risk and Reward in a Market-Based Task Service. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, Honolulu, Hawaii, USA*, 2004.

[89] S. Jang, V. Taylor, X. Wu, M. Prajugo, E. Deelman, G. Mehta, and K. Vahi. Performance Prediction-Based Versus Load-based Site Selection: Quantifying the Difference. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems, Las Vegas, Nevada*, 2005.

[90] L. Kale, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi. Faucets: Efficient Resource Allocation on the Computational Grid. In *Proceedings of the 33rd International Conference on Parallel Processing, Quebec, Canada*, 2004.

[91] W. Kang, H. Huang, and A. Grimshaw. A Highly Available Job Execution Service in Computational Service Market. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Austin, Texas, USA*, 2007.

[92] U. Kant and D. Grosu. Double Auction Protocols for Resource Allocation in Grids. In *Proceedings of the International Conference on Information Technology: Coding and Computing, Nevada, USA*, 2005.

[93] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. Cambridge University Press, 1993.

[94] T. Kelly. Generalized Knapsack Solvers for Multi-Unit Combinatorial Auctions: Analysis and Application to Computational Resource Allocation. Technical Report HPL-2004-21, HP Labs, Palo Alto, CA, USA, 2004.

[95] H. Keung, J. Dyson, S. Jarvis, and G. Nudd. Performance Evaluation of a Grid Resource Monitoring and Discovery Service. *IEE Proceedings - Software*, 150(4):243–251, 2003.

[96] A. S. Kiara Corrigan and C. Patel. Estimating Environmental Costs. In *Proceedings of the Ist USENIX Workshop on Sustainable Information Technology, San Jose, CA, USA*, 2009.

[97] K. Kim, R. Buyya, and J. Kim. Power Aware Scheduling of Bag-of-Tasks Applications with Deadline constraints on DVS-Enabled Clusters. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, 2007.

[98] S. Kim and J. Weissman. A GA-based Approach for Scheduling Decomposable Data Grid Applications. In *Proceedings of the 2004 International Conference on Parallel Processing, Montreal, Canada*, 2004.

[99] L. Kleinrock. A Vision for the Internet. *ST Journal of Research*, 2(1):4–5, 2005.

[100] L. Kleinrock and R. Gail. *Queueing Systems*. Wiley New York, 1976.

[101] S. Kumar, K. Dutta, and V. Mookerjee. Maximizing Business Value by Optimal Assignment of Jobs to Resources in Grid Computing. *European Journal of Operational Research*, 194(3):856–872, 2009.

[102] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Scheduling Jobs on the Grid–Multicriteria Approach. *Computational Methods in Science and Technology*, 12(2):123–138, 2006.

[103] Y. Kwok, S. Song, and K. Hwang. Selfish Grid Computing: Game-Theoretic Modeling and NAS Performance Results. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid, Cardiff, UK*, 2005.

[104] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman. Tycoon: An Implementation of a Distributed, Market-based Resource Allocation System. *Multiagent and Grid Systems*, 1(3):169–182, 2005.

[105] B. Lawson and E. Smirni. Power-Aware Resource Allocation in High-End Systems via Online Simulation. In *Proceedings of the 19th Annual International Conference on Supercomputing*, pages 229–238, Cambridge, USA, 2005.

[106] H. Li, D. Groep, and L. Wolters. Workload Characteristics of a Multi-Cluster Supercomputer. In *Proceedings of the 7th Internation Workshop on Job Scheduling Strategies for Parallel Processing, New York, USA*, 2004.

[107] J. Li and R. Yahyapour. Learning-based Negotiation Strategies for Grid Scheduling. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid, Singapore*, 2006.

[108] M. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, San Jose, CA, 1988.

[109] U. Lublin and D. Feitelson. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[110] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[111] J. Markoff and S. Hansell. Hiding in Plain Sight, Google Seeks More Power. *New York Times*, 14, 2006.

[112] S. Martello and P. Toth. An Algorithm for the Generalized Assignment Problem. *Operational research*, 81:589–603, 1981.

[113] S. Mingay. ITs Role in a Low Carbon Economy. *Keynote Address, Greening the Enterprise*, 2, 2008.

[114] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *Proceedings of the 2005 USENIX Annual Technical Conference, Anaheim, CA*, 2005.

[115] A. Mu'alem and D. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimatesin Scheduling the IBM SP2 with Backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.

[116] E. Munir, J. Li, and S. Shi. QoS Sufferage Heuristic for Independent Task Scheduling in Grid. *Information Technology Journal*, 6(8):1166–1170, 2007.

[117] J. Nakai and R. Van Der Wijngaart. Applicability of Markets to Global Scheduling in Grids. *NAS Report*, pages 03–004, 2003.

[118] M. Narumanchi and J. Vidal. Algorithms for Distributed Winner Determination in Combinatorial Auctions. *Agent-Mediated Electronic Commerce. Designing Trading Agents and Mechanisms*, 3937:43–56, 2006.

[119] R. Nelson and T. Philips. An Approximation to the Response Time for Shortest Queue Routing. *ACM SIGMETRICS Performance Evaluation Review*, 17(1):181–189, 1989.

[120] R. Nelson and T. Philips. An Approximation for the Mean Response Time for Shortest Queue Routing with General Interarrival and Service Times. *Performance Evaluation*, 17(2):123–139, 1993.

[121] D. Neumann, J. Stoesser, A. Anandasivam, and N. Borissov. Sorma-Building an Apen Grid Market for Grid Resource Allocation. In *Proceedings of the 4th International Workshop of Grid Economics and Business Models, Rennes, France*, 2007.

[122] N. Nisan. Bidding and Allocation in Combinatorial Auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 1–12, New York, NY, USA, 2000.

[123] S. Nozaki and S. Ross. Approximations in Finite-Capacity Multi-Server Queues with Poisson Arrivals. *Journal of Applied Probability*, 15(4):826–834, 1978.

[124] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. Pace–A Toolset for the Performance Prediction of Parallel and Distributed Systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[125] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed Resource Discovery on PlanetLab with SWORD. In *Proceedings of the 1st Workshop on Real, Large Distributed Systems, San Fancisco, CA, USA*, 2004.

[126] A. Orgerie, L. Lefèvre, and J. Gelas. Save Watts in Your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems. In *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, Melbourne, Australia, 2008.

[127] P. Padala, C. Harrison, N. Pelfort, E. Jansen, M. Frank, and C. Chokkareddy. OCEAN: The Open Computation Exchange and Arbitration Network, A Market Approach to Meta Computing. In *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing, Ljubljana, Slovenia*, 2003.

[128] C. Patel, R. Sharma, C. Bash, and M. Beitelmal. Energy Flow in the Information Technology Stack: Coefficient of Performance of the Ensemble and its Impact on the Total Cost of Ownership. *HP Labs External Technical Report, HPL-2006-55*, 2006.

[129] C. Patel, R. Sharma, C. Bash, and S. Graupner. Energy Aware Grid: Global Workload Placement based on Energy Efficiency. Technical Report HPL-2002-329, HP Labs, Palo Alto, Nov. 2002.

[130] P. Pillai and K. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles, Banff, Canada*, 2001.

[131] R. Porter. Mechanism Design for Online Real-Time Scheduling. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 61–70, New York, USA, 2004.

[132] F. Ramme, T. Romke, and K. Kremer. A Distributed Computing Center Software for the Efficient Use of Parallel Computer Systems. In *Proceedings of the 1994 International Confernece on High Performance Computing and Networking, Munich, Germany*, 1994.

[133] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A Balanced Energy-Efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2007.

[134] I. Rodero, F. Guim, J. Corbalan, and J. Labarta. eNANOS: Coordinated Scheduling in Grid Environments. In *Proceedings of the 2005 International Conference on Parallel Computing (ParCo), Malaga, Spain*, 2005.

[135] L. Rudolph and P. Smith. Valuation of Ultra-scale Computing Systems. In *Proceedings of 2000 Job Scheduling Strategies for Parallel Processing, Cancun, Mexico*, 2000.

[136] G. Sabin, V. Sahasrabudhe, and P. Sadayappan. Assessment and Enhancement of Meta-Schedulers for Multi-Site Job Sharing. In *Proceedings of 14th IEEE International Symposium on High Performance Distributed Computing, Research Triangle Park, NC*, 2005.

[137] V. Salapura et al. Power and Performance Optimization at the System Level. In *Proceedings of the 2nd Conference on Computing Frontiers*, Ischia, Italy, 2005.

[138] M. Saltzman. COIN-OR: An Open-Source Library for Optimization. *Programming Languages and Systems in Computational Economics and Finance*, page 1, 2002.

[139] T. Sandholm, K. Lai, and S. Clearwater. Admission Control in a Computational Market. In *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, France*, 2008.

[140] T. Sandholm, K. Lai, J. Ortiz, and J. Odeberg. Market-Based Resource Allocation using Price Prediction in a High Performance Computing Grid for Scientific Applications. In *Proceedings of 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France*, 2006.

[141] H. A. Sanjay and S. Vadhiyar. Performance Modeling of Parallel Applications for Grid Scheduling. *Journal of Parallel Distributed Computing*, 68(8):1135–1145, 2008.

[142] B. Schnizler. MACE: a Multi-Attribute Combinatorial Exchange. *Negotiation, Auctions, and Market Engineering*, 2:84–100.

[143] B. Schnizler. Resource Allocation in the Grid. A Market Engineering Approach, Ph.D. thesis. *Studies on eOrganisation and Market Engineering*, 2007.

[144] J. Shirazi. *Java performance tuning*. O'Reilly Media, Inc., 2003.

[145] E. Shmueli and D. Feitelson. Backfilling with Lookahead to Optimize The Packing of Parallel Jobs. *Journal of Parallel and Distributed Computing*, 65(9):1090–1107, 2005.

[146] J. Shneidman, C. Ng, D. Parkes, A. AuYoung, A. Snoeren, A. Vahdat, and B. Chun. Why Markets Could (but Dont Currently) Solve Resource Allocation Problems in Systems. In *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating System, Santa Fe, NM, USA*, 2005.

[147] G. Singh, C. Kesselman, and E. Deelman. A Provisioning Model and its Comparison with Best-Effort for Performance-Cost Optimization in Grids. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing, California, USA*, 2007.

[148] G. Singh, C. Kesselman, and E. Deelman. Adaptive Pricing for Resource Reservations in Shared Environments. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Texas, USA*, 2007.

[149] R. Smith. The Contract Net Protocol High-Level Communication and Control in a Distributed Porblem Solver. *IEEE Transaction on Computer*, 4:1104–1113, 1980.

[150] W. Smith, I. Foster, and V. Taylor. Predicting Application Run Times Using Historical Information. In *Proceedings of IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, Florida, USA*, 1998.

[151] O. Sonmez and A. Gursoy. A Novel Economic-based Scheduling Heuristic for Computational Grids. *International Journal of High Performance Computing Applications*, 21(1):21, 2007.

[152] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An Economic Paradigm for Query Processing and Data Migration in Mariposa. In *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, Austin, Texas, USA*, 1994.

[153] J. Stosser, P. Bodenbenner, S. See, and D. Neumann. A Discriminatory Pay-as-Bid Mechanism for Efficient Scheduling in the Sun N1 Grid Engine. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences, Hawaii*, 2008.

[154] G. Stuer, K. Vanmechelen, and J. Broeckhove. A Commodity Market Algorithm for Pricing Substitutable Grid Resources. *Future Generation Computer Systems*, 23(5):688–701, 2007.

[155] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with GRIA Industrial Applications on a Web Services Grid. In *Proceedings of the Ist International Conference on e-Science and Grid Computing, Melbourne, Australia*, 2005.

[156] Z. Tan and J. Gurd. Market-based Grid Resource Allocation using a Stable Continuous Double Auction. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Austin, Texas, USA*, 2007.

[157] Q. Tang, S. K. S. Gupta, D. Stanzione, and P. Cayton. Thermal-Aware Task Scheduling to Minimize Energy Usage of Blade Server Based Datacenters. In *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, Los Alamitos, CA, USA, 2006.

[158] G. Tesauro et al. Managing Power Consumption and Performance of Computing Systems Using Reinforcement Learning. In *Proceedings of the 21st Annual Conference on Neural Information Processing Systems*, Vancouver, Canada, 2007.

[159] United States Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431. `http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf`, Aug. 2007.

[160] U.S. Department of Energy. US Energy Information Administration (EIA) report. `http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_a.html`, 2007.

[161] U.S. Department of Energy. Voluntary Reporting of Greenhouse Gases: Appendix F. Electricity Emission Factors, 2007. http://www.eia.doe.gov/oiaf/1605/pdf/Appendix20F_r071023.pdf.

[162] S. Venugopal, X. Chu, and R. Buyya. A Negotiation Mechanism for Advance Resource Reservation using the Alternate Offers Protocol. In *Proceedings of the 16th International Workshop on Quality of Service, Netherlands*, 2008.

[163] S. Venugopal, K. Nadiminti, H. Gibbins, and R. Buyya. Designing a Resource Broker for Heterogeneous Grids. *Software: Practice and Experience*, 38(8):793–826, 2008.

[164] G. Verdun, D. Azevedo, H. Barrass, S. Berard, M. Bramfitt, T. Cader, T. Darby, C. Long, N. Gruendler, B. Macarthur, et al. The Green Grid Metrics: Data Center Infrastructure Efficiency (DCIE) Detailed Analysis. *The Green Grid*, 2008.

[165] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kep hart, and W. S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[166] L. Wang, H. J. Siegel, V. Royehowdhury, and A. Maciejewski. Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, 1997.

[167] L. Wang and Y. Lu. Efficient Power Management of Heterogeneous Soft Real-Time Clusters. In *Proceedings of the 2008 Real-Time Systems Symposium*, Barcelona, Spain, 2008.

[168] R. Wolski, J. Plank, J. Brevik, and T. Bryan. Analyzing Market-based Resource Allocation Strategies for the Computational Grid. *International Journal of High Performance Computing Applications*, 15(3):258, 2001.

[169] R. Wolski, J. Plank, J. Brevik, and T. Bryan. G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid. In *Proceedings of the 2001 International Parallel and Distributed Processing Symposium, CA, USA*, 2001.

[170] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15:757–768, 1999.

[171] L. Xiao, Y. Zhu, L. Ni, and Z. Xu. GridIS: An Incentive-Based Grid Scheduling. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, USA*, 2005.

[172] W. Xiaohui, D. Zhaohui, Y. Shutao, H. Chang, and L. Huizhen. CSF4: A WSRF compliant meta-scheduler. In *Proceedings of the 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing, Las Vegas, Nevada, USA*, 2006.

[173] B. A. Yair Amir and R. S. Borgstrom. The Java Market: Transforming the Internet into a Metacomputer. Technical Report CNDS-98-1, Johns Hopkins University, 1998.

[174] C. Yeo and R. Buyya. Service Level Agreement based Allocation of Cluster Resources: Handling Penalty to Enhance Utility. In *Proceedings of the 7th IEEE International Conference on Cluster Computing, Boston, USA*, 2005.

[175] C. Yeo and R. Buyya. A Taxonomy of Market-based Resource Management Systems for Utility-Driven Cluster Computing. *Software: Practice and Experience*, 36(13):1381, 2006.

[176] J. Yu, R. Buyya, and C. Tham. Cost-based Scheduling of Scientific Workflow Applications on Utility Grids. In *Proceeding of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia*, 2005.

[177] J. Yu, S. Venugopal, and R. Buyya. A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services. *The Journal of Supercomputing*, 36(1):17–31, 2006.

[178] W. Zhang, A. Cheng, and M. Hu. Multisite Co-allocation Algorithms for Computational Grid. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece*, 2006.

[179] H. Zhao and X. Li. Efficient Grid Task-Bundle Allocation Using Bargaining Based Self-Adaptive Auction. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China*, 2009.

[180] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: A Load sharing Facility for Large, Heterogeneous Distributed Computer Systems. *Software, Practice & Experience*, 23(12):1305–1336, 1993.