NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Dynamic Matching of Independent Jobs on a Computing Grid

By

Navdeep Kaur Kapoor, B.Eng. (EE)

A thesis submitted to the Faculty of Graduate Studies and Research in partial
fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
December 2005

© Copyright 2005, Navdeep Kaur Kapoor

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-494-13442-9

Our file Notre référence

ISBN: 0-494-13442-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Grid Computing has emerged as a new paradigm for distributed computing. It promotes sharing of distributed resources that may be heterogeneous in nature and enables scientists and engineering professionals to solve large scale computing problems. With grid computing, businesses can efficiently utilize computing and data resources and combine them for large capacity workloads. The primary benefit of grid computing is the ability to coordinate and share resources.

To maximize the benefits of grid computing, it is essential to be able to discover the resources available on the grid, and also be able to map the jobs to the resources most effectively to maximize the given objective function. In computational grids, the problem of resource discovery and mapping (matching and scheduling) of jobs to resources is really challenging due to the large number of resources, resource heterogeneity and variation in resource attributes such as CPU load, network bandwidth etc. This research focuses on the problem of dynamic matching of jobs to resources in a computing grid. The state of the art resource discovery and dynamic mapping policies are studied in detail. Most of the mapping policies assume complete knowledge about the characteristics of jobs and resources while making an allocation decision. However, this may not be feasible in reality. This research introduces dynamic matching policies based on job classes, which match jobs to resources on a grid without any detailed knowledge about the characteristics of jobs and resources. Under a variety of different workload parameters they demonstrate a performance comparable to or better than the minimum completion time mapping heuristic that is based on exact *a priori* knowledge of job and resource characteristics.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my co-supervisors Professor Shikharesh Majumdar and Professor Biswajit Nandy for their invaluable guidance, continuous support, and numerous encouragements during the thesis.

I would also like to thank Anthony Sulistio, from University of Melbourne for his prompt support regarding the simulation toolkit whenever it was needed.

Also, I would like to express my deep appreciation and thanks to my husband for his patience and strong support.

Last but not the least, I want to thank my twenty-month-old twins who could not get the time, they deserved and needed the most due to my involvement in the research.

Table of Contents

1. INTRODUCTION	1
1.1 Background.....	1
1.2 Types of Grids.....	3
1.2.1 Computing Grid	3
1.2.2 Scavenging Grid.....	3
1.2.3 Data Grid.....	3
1.3 Evolution of Grid	4
1.3.1 Clusters.....	4
1.3.2 Intra-Grids.....	4
1.3.3 Extra-Grids.....	4
1.3.4 Inter-Grid.....	5
1.4 Grid Components: A High Level Perspective	5
1.4.1 Security	6
1.4.2 Broker.....	6
1.4.3 Scheduler.....	7
1.4.4 Data Management	7
1.4.5 Job & Resource Management	8
1.5 Motivation for the Thesis.....	8
1.6 Goals of the Thesis.....	9
1.7 Contributions of the Thesis.....	9
1.8 Thesis Outline	10
2. OVERVIEW OF RESOURCE DISCOVERY AND MAPPING IN A GRID	11
2.1 Resource Discovery in a Grid.....	11
2.1.1 Elements of Resource Discovery in a Grid.....	11

2.1.2	Algorithms for Data Dissemination for Resource Discovery	13
2.2	Mapping in a Grid	15
2.2.1	Online Matching of a Class of Independent Tasks	17
2.2.2	Heuristics for Batch Mode Mapping.....	20
2.2.3	Workqueue with Replication (WQR)	24
2.2.4	RR Scheduling	26
2.2.5	Commercially Available Scheduling Systems	27
3.	THE SIMULATION MODEL.....	30
3.1	System Architecture.....	30
3.2	The GridSim Toolkit.....	31
3.2.1	Architecture of GridSim.....	31
3.3	Simulation Model.....	32
3.3.1	Source.....	33
3.3.2	Broker.....	33
3.3.3	Regional Grid Information Service.....	34
3.3.4	Resource.....	34
3.3.5	Router.....	34
3.3.6	Links.....	35
3.4	Dynamic Matching Policies.....	35
3.4.1	Minimum Completion Time	36
3.4.2	Class Based	37
3.4.3	Class Based Round Robin.....	38
3.4.4	Class Based with Load Information.....	39
3.5	Performance Measures.....	40
3.5.1	Mean Response Time (R_t).....	40

3.6	Parameters.....	40
3.6.1	System Parameters	40
3.6.2	Workload Parameters	42
3.7	Verification of Simulation Experiments	45
3.7.1	Little's Law	46
3.8	Accuracy of Measurement	46
3.8.1	Initial Bias	46
3.8.2	Test Duration.....	47
4.	PERFORMANCE OF CLASS BASED MATCHING ALGORITHMS	48
4.1	Objectives	48
4.2	Performance of the Class Based Algorithm.....	49
4.3	Effect of Number of Clusters.....	51
4.3.1	Performance of Class-Based Matching in Three Clusters	51
4.3.2	Performance of Class-Based Matching in Four Clusters	53
4.3.3	Performance of Class-Based Matching in Six Clusters	56
4.4	Effect of Number of Resources in Each Cluster.....	58
4.4.1	Performance with Two Resources in Each Cluster.....	58
4.4.2	Performance with Four Resources in Each Cluster.....	62
4.5	Effect of Number of Classes of Jobs.....	64
4.6	Effect of Coefficient of Variation of Job Demands	68
4.7	Effect of Mean Service Demand of Jobs	74
4.8	Effect of Communication Delays.....	75
4.9	Class Based Full Round Robin Algorithm.....	75
5.	CONCLUSIONS	79
5.1	Summary	79

5.2	Future Research	82
REFERENCES.....		84

LIST OF TABLES

Table 3-1: Default Values of System & Workload Parameters	45
Table 4-1: System & Workload Parameters to study the effect of number of classes on Mean Response Time.....	65
Table 4-2: Workload Parameters for various coefficient of variation	69

LIST OF FIGURES

Figure 1-1: Evolution of Grid Computing	5
Figure 1-2: The Primary Components of a Grid	6
Figure 2-1: Mapping Algorithms	16
Figure 3-1: Architecture of GridSim.....	32
Figure 3-2: The Simulation Model	33
Figure 4-1 : Performance of class based matching for $N_c=2$	51
Figure 4-2: Mean Response Time of jobs for $N_c=3$	53
Figure 4-3: Mean Response Time of jobs at all loads for $N_c=4$	55
Figure 4-4: Mean Response Time of jobs at low & medium load for $N_c=4$	56
Figure 4-5: Mean Response Time of jobs at all loads for $N_c=6$	57
Figure 4-6: Mean Response Time of jobs at low and medium load for $N_c=6$	57
Figure 4-7: Mean Response Time of jobs in configuration one for $N_r=2$	60
Figure 4-8: Mean Response Time of jobs in configuration two for $N_r=2$	61
Figure 4-9: Mean Response Time of jobs at all loads for $N_r=4$	63
Figure 4-10: Mean Response Time of jobs at low & medium load for $N_r=4$	63
Figure 4-11: Mean Response Time of jobs at all loads for $N_{classes}=3$	66
Figure 4-12: Mean Response Time of jobs at low & medium load for $N_{classes}=3$	66
Figure 4-13: Mean Response Time of jobs for C.O.V=1.09	70
Figure 4-14: Mean Response Time of jobs for C.O.V=1.5	70
Figure 4-15: Mean Response Time of jobs for C.O.V=2.94	71
Figure 4-16: Mean Response Time of jobs for C.O.V = 3.41	72
Figure 4-17: Performance of various policies for different values of COV at $\lambda=0.02$	73
Figure 4-18: Performance of various policies for different values of COV at $\lambda=0.0275$. 73	
Figure 4-19: Effect of M_{SD} on Mean Response Time.....	74

Figure 4-20: Performance of CB-FRR in configuration one for $N_r = 2$	76
Figure 4-21: Performance of CB-FRR in configuration two for $N_r = 2$	77

GLOSSARY OF TERMS AND SYMBOLS

λ	Arrival rate of jobs
CB	Class Based
CB-FRR	Class Based Full Round Robin
CB-LI	Class Based with Load Information
CB-RR	Class Based Round Robin
COV	Coefficient of Variation of Job Demands
D_d	Delay due to discovery at GIS
D_p	Propagation Delay
D_t	Transmission Delay
GASS	Grid Access to Secondary Storage
GIS	Grid Information Service
GRAM	Grid Resource Allocation Manager
GSI	Grid Security Infrastructure
KPB	K Percent Best
LA	Local Allocation
LDAP	Lightweight Directory Access Protocol
MCT	Minimum Completion Time
MDS	Monitoring and Directory Service
MET	Minimum Execution Time
MI	Million Instructions
M_{SD}	Mean Service Demand
N_c	Number of clusters
$N_{classes}$	Number of classes
N_r	Number of resources in a cluster

GLOSSARY OF TERMS AND SYMBOLS

OLB	Opportunistic Load Balancing
PDP	Prioritized Dissemination Protocol
p_l	Probability of job in class LARGE
p_m	Probability of job in class MEDIUM
p_s	Probability of job in class SMALL
QoS	Quality of Service
RR	Round Robin
R_t	Mean Response Time
SA	Switching Algorithm
VO	Virtual Organization
WQR	Workqueue with Replication

Chapter 1: Introduction

1.1 Background

The concept of grid first emerged in mid-1990s. However, it was only around 2000, that grids moved beyond the domain of the research community and different organizations started adopting the grid to support various computing applications. Today, use of grid computing can be seen in larger businesses, for demanding applications and department specific processes [31].

The term grid was coined as an analogy with the power grid. The power grid is responsible for supplying consistent, dependable, and transparent access to an electrical supply. Likewise, grid computing is intended to provide an equally consistent, dependable, and transparent collection of computing resources.

The grid denotes a distributed computing infrastructure for advance science and engineering [13]. Within organizations grids support scientific and technical computing applications. Grid Computing provides direct access to computers, software, and storage devices. Conventional networks focus on communication among devices. “The vision of grid computing is to virtualize computing, with the goal of creating a utility computing model over a distributed set of resources” [36]. A stand-alone computer comprises of elements like a processor, storage, operating system, and I/O. The concept of grid computing is to create this environment on a large scale in a distributed area. Similar to a processor, storage, operating system, and I/O in a computer, a grid comprises of servers, storage devices, and networks. The software responsible for coordination of various elements participating in the grid is similar to the operating system in a computer.

Ian Foster, head of the Distributed Systems Laboratory at Argonne National Laboratory, has given a simple three-point checklist to define a grid [17]. He has defined a grid as a system that,

1. “Coordinates resources that are not subject to centralized control.”
2. “Uses standard, open, general-purpose protocols and interfaces.”
3. “Delivers nontrivial qualities of service.”

In other words, grid computing is a way to share computing resources within and among organizations. The resources in a grid may be identified as computational resources, storage resources, network resources, and code repositories [13]. The resources are a part of a “virtual organization” [15], which may comprise of several individuals and institutions. The resources in a virtual organization are connected via a network (possibly the internet) and may be owned by different organisations or individuals and are shared under locally defined policies. The policies define who is allowed to share what and under what conditions. The middleware layer in a grid provides the services for security, monitoring of jobs, and accessing information.

In the research presented in this thesis, only computational resources are considered. In computing grids, unused processing cycles of multiple computers in a network are used for solving problems that would be intensive for a stand-alone computer. This leads to higher utilisation of resources.

Computing grids differ from other distributed systems by way of their dynamic nature. The resources join or leave the pool. In addition, the resource attributes like CPU load, available bandwidth keep on varying over time.

Various classes of grids are discussed in the next section.

1.2 Types of Grids

Grids have been categorized from the application perspective and as well as from the topology perspective. With respect to the application perspective, grids have been categorized based on the solutions they best address. The categories of grids as defined in [26] are explained in this section.

1.2.1 Computing Grid

A computing grid is a collection of computing resources, set aside specifically for achieving high computing power. In this type of grid, most of the machines are high-performance servers. The resources are aggregated to act as a unified processing resource. The resources may be a part of the same or different administrative domains. As mentioned in the previous section, the research in this thesis focuses on computing grid.

1.2.2 Scavenging Grid

A scavenging grid is most commonly used with a large number of desktop machines. Machines are scavenged for available CPU cycles and other resources. Owners of the desktop machines are usually given control over when their resources are available to participate in the grid.

1.2.3 Data Grid

A data grid is responsible for housing and providing access to data across multiple organizations. The users have access to the data but the location of the data is transparent to the user. A data grid allows organizations to share their data, manage the data, and

manage security issues such as who has access to what data. Data grids enable users and applications to have access to data across distributed locations.

1.3 Evolution of Grid

With respect to the topology perspective, the grids have been classified into clusters, intra-grids and extra-grids [36]. However, it is debated that these classifications denote the stages of evolution of a grid, and in each of these stages it is possible to classify the grid based on the solution it addresses: computing grid, scavenging grid or a data grid.

1.3.1 Clusters

Clusters are critical to the evolution of a grid. Clusters are aggregation of servers in order to provide increased computing power as compared to stand-alone computers.

1.3.2 Intra-Grids

Intra-Grids comprise of inter-connected clusters. Connecting clusters enables the creation of enterprise or interdepartmental grids. With intra-grids, issues relating to interdepartmental authentication and security need to be addressed.

1.3.3 Extra-Grids

Interconnected cluster grids and/or intra-grids referred to as Extra-Grids are geographically distributed between enterprise organizations. They are also referred to as partner grids. As the enterprise organizations are geographically apart, network latency needs to be considered.

1.3.4 Inter-Grid

An Inter-Grid is a wide area service grid. Instead of two or three organizations participating in a grid, like in an extra-grid, an inter-grid could serve thousands of users and organizations. This is the final stage of the grid evolution [36]. Though, commercially we are not there yet, these grids do exist in the research and development field.

Various stages of evolution of grid computing are shown in Figure 1-1 [36].

Figure 1-1: Evolution of Grid Computing

1.4 Grid Components: A High Level Perspective

Globus Alliance is an association dedicated to develop fundamental technologies that are needed to provide a secure and reliable computing infrastructure in a grid. Globus Toolkit [14] consists of software components available under an open-source license

from Globus Alliance for grid computing. The primary components of a grid as described in [26] are shown in Figure 1-2 and described in the following sub-sections.

Figure 1-2: The Primary Components of a Grid
(Figure taken from [26])

1.4.1 Security

It has to be ensured that only authorized users use the grid resources and services. In the Globus Toolkit [14], the security mechanisms are provided by the Grid Security Infrastructure (GSI) [16] component. The GSI component provided by the Globus toolkit provides functionalities such as mutual authentication, and single sign-on.

1.4.2 Broker

The broker performs the task of matching jobs to resources. Based on the job submitted by the user, the broker locates an appropriate resource in the grid that can execute the job. The broker queries a grid information service to get information regarding the available resources [10]. Globus [14] provides an LDAP [22] based information service called the Monitoring and Directory Service.

1.4.3 Scheduler

If various jobs in an application require inter-process communication and need to be scheduled at the same time, advance reservation of resources may be required for co-allocation of the jobs. Separate schedulers will be required to co-ordinate the scheduling of jobs at resources in such cases. There could be several levels of schedulers in a grid environment. A meta-scheduler can exist at a higher level and interacts with schedulers at cluster level (each cluster may have its own scheduler that coordinates the scheduling of jobs on resources belonging to the local cluster) to schedule a meta-job that comprises of several jobs, which need to be scheduled at the same time. The (synchronous) resource allocation and job scheduling in a grid environment is referred to as metascheduling. Globus does not provide schedulers. However, many commercially available schedulers can be used in a Globus environment. Some of these schedulers are described in Section 2.2.5.

1.4.4 Data Management

For transfer of data within the grid environment, there needs to be a secure and reliable method for moving files and data to various nodes within the grid. The Globus Toolkit contains a Grid Access to Secondary Storage (GASS) [4] component, which is a data management component to provide such services. This component includes facilities such as the GridFTP [2]. GridFTP is built on top of the standard FTP protocol. It adds additional functionality in the context of the grid. It utilizes the GSI [16] for user authentication and authorization. If a user has been authenticated once, he can use GridFTP to transfer files within nodes.

1.4.5 Job & Resource Management

The Grid Resource Allocation Manager (GRAM) [11] provided by Globus is responsible for launching a job on a particular resource. This is one of the most important services in a grid environment. It checks the status of the job and retrieves the results when it is complete.

1.5 Motivation for the Thesis

Discovery of appropriate resources and dynamic mapping of jobs to the discovered resources is one of the most important and basic services in a grid environment. Resource discovery is a process responsible for generating a set of best possible candidates for the given set of attributes. As discussed in Section 1.4.2, the broker locates an appropriate resource in the grid based on the requirements or attributes of the resource defined by the user. Dynamic mapping of jobs to resources comprises of two parts: matching and scheduling [5]. Matching refers to the process of assigning a job to a particular resource. It is a part of resource management mechanism to allocate best resource(s) from the discovered set based on some criterion. The broker as discussed in Section 1.4.2 usually performs this function. Scheduling refers to the order in which the jobs assigned to a particular resource execute (when multiple jobs contend for a resource). In literature, mapping is often referred to as scheduling.

The resource discovery and matching of jobs to resources in a grid becomes challenging due to the following reasons:

1. Large number of resources.
2. Heterogeneity in resource types and in types of user requests.

3. Dynamic nature of the grid environment with individuals and institutions joining and leaving the grid.
4. The changing nature of resource characteristics like availability and CPU load.
5. Diversity in sharing characteristics of the resources. For example, resources may be space shared (run the executing job to completion before beginning the new job). Resources could also be time shared (multiplex the execution of allocated jobs).
6. The resources may use different local resource allocation policies.

1.6 Goals of the Thesis

This thesis focuses on the problem of dynamic matching of jobs to resources in a grid. The state of the art resource discovery and matching algorithms are studied and new dynamic matching algorithms are introduced. Scheduling of jobs on the matched resources is assumed to follow a first come first served (FCFS) policy. The performance of the proposed algorithms is compared to the minimum completion time mapping algorithm [29] by running a series of simulation experiments.

1.7 Contributions of the Thesis

The main contributions of the thesis are:

- Introduction of a dynamic matching technique, which does not require detailed *a priori* knowledge about the characteristics of jobs and resources. With the proposed algorithms, a class is associated with each job based on the service demand of the job. A class is also associated with each resource. Jobs of a particular class are submitted to a resource that is designated to perform the jobs

of that class.

- Evaluation of the performance of the proposed algorithms by running a series of simulation experiments in which system and workload parameters such as the number of clusters, the number of resources in a cluster, the number of classes of jobs, the arrival rate of jobs, as well as the coefficient of variation and mean service demand of jobs are varied. The results of the experiments lead to insights underlying matching of jobs to resources and system performance.
- The performance of the class-based matching algorithms is compared to the well-known mapping heuristic: minimum completion time (MCT) [29]. This heuristic assumes complete knowledge of characteristics of jobs and resources. If the resources are well mapped to classes, class-based matching in most cases provides performance comparable to the minimum completion time algorithm. In some cases, class-based matching even outperforms MCT.

1.8 Thesis Outline

The thesis comprises of five chapters. Chapter 2 provides an overview of the state of the art resource discovery and matching policies in a grid. Chapter 3 gives details of the simulation model and the simulation toolkit used. It presents the various proposed matching policies in detail. It also provides an overview of the system parameters, workload parameters, performance measures and test conditions. Chapter 4 describes the different simulation experiments performed in detail. Effect of various system and workload parameters on the performance of the matching policies is discussed. Chapter 5 concludes this thesis and discusses directions for future research.

Chapter 2: Overview of Resource Discovery and Mapping in a Grid

This chapter presents the state of the art resource discovery and matching policies in the grid environment. Resource discovery issues in a grid are discussed in Section 2.1.1 and some algorithms for non-uniform information dissemination are discussed in Section 2.1.2. Both immediate and batch mode mapping policies in a distributed computing environment are studied in Section 2.2.1 and Section 2.2.2 respectively. Some algorithms that perform matching decisions based on no information about characteristics of job and resources are discussed in Section 2.2.3 and Section 2.2.4. Finally Section 2.2.5 presents some commercially available grid schedulers.

2.1 Resource Discovery in a Grid

Grid technologies enable sharing of large number of resources that are a part of a “virtual organisation” [15]. Resource discovery is a process responsible for generating a set of potential resources for the given set of attributes [27]. Resource discovery in a grid is a challenging task due to the large number of resources, their dynamic behaviour, diverse nature and geographical distribution of the resources.

2.1.1 Elements of Resource Discovery in a Grid

As described in [24], the principles that guide the design of a resource discovery mechanism include the following:

- No centralized control: Due to the large number of resources and their dynamic nature, it is not possible for an individual or an institution to manage such a large number of resources.

- The resources are diverse in nature, therefore structures assuming homogeneity in resources and their behavior should be avoided.
- It is not possible to impose a global naming scheme.

The authors in [24] describe four sub problems that define a general resource discovery solution. They call it the four axes of the solution space. These sub problems are described next.

2.1.1.1 Membership Protocol

The grid is a dynamic environment with nodes joining and departing the system. The membership protocol refers to how new nodes join the network and how nodes learn about others in the network.

2.1.1.2 Overlay Construction Function

Due to the large size of the grid, the number of nodes to be queried may be limited by available bandwidth, message processing load, security or administrative policies, and topology specifications. The Overlay Construction Function selects the set of active collaborators/nodes from the local membership list.

2.1.1.3 Pre-processing/ Offline Processing

Due to the large scale of the grid, offline preparations are made for better performance. One-way of pre-processing is to disseminate resource descriptions i.e. advertise description of local resources to other areas of the network. Some algorithms for achieving this are discussed in Section 2.1.2.

2.1.1.4 Request Processing

Request processing can have two components: local processing and remote processing. Local processing refers to the part of looking for the requested resource in the local cluster. Remote processing refers to the rules for propagating the request to other peers through various mechanisms when the request for the resource cannot be met locally. In the system described in [24], four types of remote request propagation strategies are discussed: Random walks, Learning based, Best-Neighbor rule, and Learning based + Best Neighbor rule. In case of Random walks, if a request cannot be met locally, the node to which request is forwarded is chosen randomly. In case of the Learning based strategy, the request is forwarded to the peer that satisfied similar requests previously. The nodes record the requests satisfied by other nodes. If such information does not exist, the remote node to which the request is to be propagated is chosen randomly. In case of the Best Neighbor rule, unlike the learning based propagation strategy, the type of request is not taken into account. The request is forwarded to the peer who satisfied the largest number of requests. Learning based + Best Neighbor rule is similar to the learning based propagation strategy. However, if information is not available about the types of requests satisfied by different nodes, the request is propagated to the best neighbour (the node that has satisfied the maximum number of requests).

2.1.2 Algorithms for Data Dissemination for Resource Discovery

As discussed in Section 2.1.1.3, offline preparations are made for better performance (referred to as pre-processing). One-way of pre-processing is to disseminate

resource descriptions i.e. advertise description of local resources to other areas of the network. Generally, resource discovery schemes maintain and query a resource status database. For scalable implementations, it is essential to organize the status databases in a distributed fashion. To keep the databases consistent, dissemination of resource status information is one of the key operations and the major aim is to reduce the data dissemination overhead and costs. In [27], various algorithms for data dissemination for resource discovery are discussed: Universal Awareness, Neighbourhood Awareness, and Distinctive Awareness algorithm. In the Universal Awareness algorithm, each node distributes status information uniformly and each node learns about every other node in the grid. In the Neighbourhood Awareness algorithm, the information from one node is propagated to other nodes that are less than a fixed distance away from it. This reduces dissemination overhead in large networks. In the case of Distinctive Awareness algorithm, the extent to which the information is distributed depends on the attributes of a node. If nodes have distinct attributes, the information may need to be disseminated to all nodes, like in the case of Universal Awareness algorithm. If the nodes are homogeneous, information regarding the node may be transmitted only to the neighbouring nodes, and this algorithm reduces to the neighbourhood awareness algorithm. Some approaches for non-uniform information dissemination for dynamic grid resource discovery are described in [25]: Probabilistic Protocol, Prioritized Dissemination Protocol (PDP), and Change-Sensitive Protocol. These approaches use a certain protocol to limit the dissemination of information about resources in a node to other nodes in the grid. The probabilistic protocol described in [25] is based on the fact that it is more important for a node to have accurate and fresh information about nearby resources because the overhead

of transferring data and results increases with the distance to the resources. In case of probabilistic protocol, forwarding probabilities decrease, as the node gets farther away from the resource. The PDP protocol described in [25] forwards resource information depending on the nature of the resource. If the usage of a resource/service needs to be commercialized, fresh information about that resource could lead to it being selected more frequently. If the resources are not uniformly loaded along the grid, the grid system could sense this and propagate information about the relatively lightly loaded resources to more distant repositories, to attract more usage and a better distribution of load. Also, resources may be categorized into different classes, and a different forwarding policy may be applied to each class. For example, information about scarce and more powerful resources may be forwarded to all nodes. Another protocol described in [25] is the change-sensitive protocol. In this case, volatility of disseminated information affects how aggressively it should be forwarded. Resource state such as CPU load, available memory and storage can change frequently during times of heavy usage. The forwarding probability depends on the difference between the information that might be forwarded, and the most recent information that was forwarded. The information will be forwarded only if there has been a substantial change in the resource state.

2.2 Mapping in a Grid

This section describes some algorithms for mapping (matching and scheduling) of independent tasks on heterogeneous distributed computing systems. The mapping algorithms studied are represented in Figure 2-1. As can be seen in Figure 2-1, the mapping algorithms may be categorized into static algorithms and dynamic algorithms. In

case of static mapping, the set of tasks to be mapped to resources is known *a priori* and the assignment of tasks to resources is done before the execution of the application actually begins in order to minimize the overheads of taking mapping decisions at run time, and thus minimizing the communication and execution time. However due to the dynamic nature of the grid with jobs continually arriving into the system, and resources joining or leaving the grid, static mapping may not be feasible. In case of dynamic mapping, the mapping decisions are taken as and when the jobs arrive into the system.

Figure 2-1: Mapping Algorithms

The dynamic mapping algorithms that have been studied as a part of this thesis have been categorised into knowledge based and knowledge free algorithms. The knowledge based algorithms take mapping decisions based on detailed *a priori* knowledge of characteristics of jobs and resources. The knowledge free mapping algorithms take mapping decisions without any knowledge of characteristics of jobs and resources. In the category of knowledge free algorithms, workqueue with replication and round robin with replication are described in Section 2.2.3 and Section 2.2.4 respectively.

The knowledge based mapping policies are further classified into immediate mode and batch mode and are discussed in Section 2.2.1 and Section 2.2.2 respectively. Some commercially available grid schedulers are discussed in Section 2.2.5.

Three phases have been defined for scheduling jobs on a grid [34]. It may be recalled that the term scheduling is often used loosely and comprises of both matching and scheduling of jobs on resources. The term scheduling as used here includes resource discovery, matching of jobs to resources and scheduling the job on the allocated resources. In phase one, the resources available on the grid are discovered. This usually involves querying an information service to generate a list of potential resources. In phase two, the job is matched to a particular resource (amongst the discovered resources) that meets the job requirements in the best possible way. In phase three, the submitted job is actually executed on the resource. Matching of a job to a resource performed in the second phase is a NP-complete problem [12]. Many heuristics have been proposed to obtain the match between the job requirements and the available resources. The work in this thesis focuses on this matching phase. New algorithms for dynamic matching of jobs to resources are introduced. In [29], some online algorithms for matching and scheduling of a class of independent tasks onto heterogeneous computing systems are discussed. Heuristics for immediate matching and batch mode mapping are discussed in the next two sections.

2.2.1 Online Matching of a Class of Independent Tasks

In online or immediate mode matching, a task is matched to a machine/ resource as soon as it arrives. Each task is considered only once for matching and the matching is

not changed, once it is computed. A number of immediate matching heuristics are described in the following sub-section.

2.2.1.1 Minimum Completion Time

As discussed in [29], the minimum completion time (MCT) algorithm assigns each task to the machine that results in the task's earliest completion time. Tasks may get assigned to machines that do not have the fastest execution time for the task. All machines/resources are checked and the completion time is computed based on the execution time of the task on the machine and the ready time of the machine. The ready time of the machine is the wall clock time at which the machine would complete executing the tasks already assigned to it and would be ready to compute the current task. This algorithm has been used as a benchmark for the on-line matching heuristics.

2.2.1.2 Minimum Execution Time

As discussed in [29], the minimum execution time (MET) algorithm assigns each task to the machine that produces the minimum execution time for the task. It does not consider machine ready times. It may cause severe imbalance of load across machines.

2.2.1.3 Switching Algorithm

As mentioned earlier, the MET heuristic can cause severe imbalance of load across machines by assigning more tasks to some machines than others. On the other hand, the MCT heuristic tries to balance the load by assigning tasks based on earliest completion times. The switching algorithm (SA) [29] uses MCT and MET in a cyclic fashion depending on the load distribution across the machines. A load balance index is calculated by considering the maximum ready time and minimum ready time over all

machines. The load balance index is the ratio of the minimum ready time and the maximum ready time over all machines. The load balance index can have any value between $\{0, 1\}$. If the ratio is 1, the load is evenly balanced across machines. If the ratio is 0, at least one machine is idle. Two threshold values are considered (low and high). Initially the load balance index is set to zero. The MCT heuristic is used to map the tasks until the load balance index increases to at least high. After this threshold is reached, the MET algorithm is used. When the load balance index is at or below low, the SA heuristic switches back to MCT and this cycle continues.

2.2.1.4 The K-Percent Best Heuristic

The k-percent best (KPB) heuristic [29] considers only a subset of machines while mapping a task. The subset is based on the execution times for the task on the machines. The execution time of the task on all machines is known. The machines are sorted in order of increasing execution time of the tasks. The first k% machines from this ordered set of machines are considered while making the matching decision. The task is then submitted to the machine in the formed subset that gives the minimum completion time for the task. This algorithm avoids allocating the current task on a machine which might be more suitable for a yet to arrive task. This heuristic has an advantage over the MCT algorithm as MCT lacks in this foresight about task heterogeneity. MCT might assign a task to a machine for marginal improvement in completion time, thus depriving subsequent better-suited tasks for that machine and thus increasing the overall make span as compared to the k-percent best algorithm.

2.2.1.5 The Opportunistic Load Balancing Heuristic

The opportunistic load balancing (OLB) heuristic [29] assigns task to the machine that becomes ready next. If multiple machines become ready or are available to start with, the machine is chosen arbitrarily. The OLB heuristic does not consider the execution time of the task on the machine.

2.2.1.6 Evaluation of Online Heuristics

The performance of the online heuristics discussed in the previous sub-sections is evaluated in [29]. Makespan has been used as a measure of performance. Makespan is the time when the last finishing task of the application completes execution. Both KPB and SA have elements of MCT and MET in their operation. However, only KPB attempts to combine the objectives of MET and MCT simultaneously. The KPB algorithm gives a better makespan, unless there is a fixed set of machines, which is not among $k\%$ best for any task. These machines will have a lot of idle time, thus leading to poor performance of the algorithm as compared to MCT.

2.2.2 Heuristics for Batch Mode Mapping

In batch mode mapping, the arriving tasks are not mapped as soon as they arrive as in the case of immediate mode heuristics described in Section 2.2.1. The tasks are collected in a set and are examined for mapping at prescheduled times. The set of independent tasks considered for mapping at the mapping event is called a meta-task. The meta-task may include the newly arrived tasks and the tasks that were considered for mapping at the last mapping event but did not begin execution [28]. The meta-tasks are

mapped after predefined intervals. There are two ways to determine the mapping intervals; the regular time interval strategy and the fixed count strategy [28].

In the regular time interval strategy, the meta-tasks are mapped to resources at predefined intervals of time irrespective of arrival of tasks into the system or departure of tasks after execution (for example, every 25 seconds). This type of mapping may be redundant if no new tasks have arrived or no tasks have finished execution since the last mapping event.

In the fixed count strategy, the meta-tasks are mapped when either of the following two conditions is met.

- a) The arriving task makes the size of the set of meta-tasks greater than a predetermined number.
- b) All tasks have arrived and a task completes, and the number of tasks, which are yet to begin, are greater or equal to a predetermined arbitrary number.

The batch mode heuristics discussed in [29] iteratively assign tasks to processors by considering all tasks that have not started execution. The completion times of all such tasks are computed on all the machines. At first the tasks are matched to the machines that give the earliest completion time for the task. If more than one task is matched to the same machine, the tasks are scheduled based on the mapping heuristic. For example, in case of min-min heuristic discussed in Section 2.2.2.1, the task that has the minimum earliest completion time (the task that changes the machine ready time by the least amount) is scheduled first. In other words, the task satisfying the defined objective function (based on the minimum completion times) is given priority and is scheduled

first. Some of the batch mode heuristics described in [29] are presented in the following sub-sections.

2.2.2.1 Min-Min

The min-min [29] algorithm iteratively assigns tasks to processors by considering all tasks not scheduled. Each task is tentatively scheduled on each resource. The machine that gives the earliest completion time for the task is determined. The task that has the minimum earliest expected completion time is determined and allocated to the corresponding machine. In other words, this heuristic schedules the task with the minimum MCT on the host that completes it the fastest.

2.2.2.2 Max-Min

The max-min algorithm [29] iteratively schedules the task with the maximum completion time on the host that completes it the fastest. Once the machine that provides the earliest completion time for each task is found, the task that has the maximum earliest completion time is determined and assigned to the corresponding machine. The max-min heuristic will perform better where there is more number of shorter tasks than longer ones. The resulting make span may just be determined by the execution time of the long task.

2.2.2.3 Sufferage

The sufferage heuristic [29] assigns a task to a machine that would “suffer” the most in terms of earliest completion time if a particular machine were not assigned to it. The sufferage value is defined as the difference between the earliest completion time and

second earliest completion time of a task. The task that has the maximum sufferage value is given priority in scheduling.

As discussed in [29], the batch mode heuristics may cause starvation of some tasks. At each mapping event, the newly arrived tasks and the tasks that are yet to begin execution are considered. Therefore, some tasks may be remapped at successive mapping events without actually beginning execution. To reduce starvation, an aging factor may be associated with a task. The aging factor will increase the probability of older tasks beginning before the tasks that would otherwise begin first. The value of earliest completion time obtained may be multiplied by the aging factor to incorporate the effect of aging.

2.2.2.4 Evaluation of Batch Mode Heuristics

The performance of the batch mode heuristics described earlier is compared in [29]. The sufferage heuristic considers the delay in completion of a task if it is not allocated to a resource/machine that gives it the earliest completion time. The sufferage heuristic performed better than the min-min and the max-min heuristics, although it performed only slightly better than the min-min heuristic.

2.2.2.5 QoS based Grid Scheduling Model

A QoS guided scheduling algorithm for grid computing was introduced in [20]. The algorithms discussed earlier do not consider QoS requirements of tasks. QoS may have a different meaning for different types of resources. Different tasks may have different QoS requirements. For a network, the QoS required may mean the desirable bandwidth for the application. For computing resources, the QoS required may mean the

CPU speed or CPU utilization. In [20], QoS is associated with each task and the scheduling algorithm is guided by the min-min heuristic. Instead of mapping the whole meta-task to the host, the tasks with high QoS are mapped first.

All the algorithms discussed in Section 2.2.1 and Section 2.2.2 assume complete knowledge about the characteristics and resources. In the next two sections, algorithms that perform matching based on no information about characteristics of jobs and resources are presented.

2.2.3 Workqueue with Replication (WQR)

In [32], the authors deal with the scheduling problem in a grid. Their work is based on scheduling applications, whose tasks are completely independent like parameter sweep applications. The independent tasks are referred to as a “bag of tasks”. However, the authors argue that good scheduling decisions always require complete knowledge about characteristics of jobs and resources that may not always be feasible. They have introduced a “knowledge free” scheduling algorithm, called the workqueue with replication.

This algorithm is an extension of the classic workqueue algorithm [32]. In the classic workqueue algorithm, the tasks are chosen arbitrarily from the bag of tasks and allocated to processors as and when they become available. However, the workqueue scheduling does not give performance comparable to scheduling algorithms that have complete knowledge about tasks and resources. The main disadvantage of workqueue scheduling is that if a large task gets allocated to a slow machine towards the end of the schedule, this could increase the overall makespan of the application. The workqueue

with replication scheduling adds replication to the classic workqueue scheduling. In the beginning, this algorithm works similar to the workqueue algorithm and allocates tasks to processors arbitrarily until the “bag of tasks” becomes empty. Whenever a processor becomes free, in case of the original workqueue scheduling, it would stay idle when all tasks have been scheduled. However, in case of workqueue with replication scheduling, the processors that become free are allocated replica of tasks that are still running. The tasks are replicated up to a predefined maximum number of replicas. Whenever a replica is allocated to a host, there is a greater possibility that it will get allocated to a fast resource, as there is a greater possibility of faster resources getting idle earlier. Whenever a task replica finishes, the other replicas are aborted or cancelled. The main disadvantage of this approach is that it wastes a lot of CPU cycles (consumed by the replicas that are cancelled).

The performance of workqueue with replication scheduling is compared to classic workqueue scheduling, dynamic fastest processor to largest task first (DFPLTF), and sufferage. The sufferage algorithm has been discussed earlier in this chapter. In DFPLTF scheduling, the tasks are sorted by size and in a descending manner. The available time of all hosts is set to zero. The largest task is allocated to the host that provides the best completion time for the task. The tasks are allocated until all machines have been assigned tasks, after which the application begins. Whenever a task finishes, all tasks that are not running are unscheduled and rescheduled again until all the machines become busy again. The process continues till all tasks finish execution. This strategy takes into account the dynamicity of the grid by rescheduling the tasks whenever a task finishes execution.

The main disadvantage of workqueue with replication is that it wastes a lot of CPU cycles, however the authors recommend that replication beyond two levels not be used and that replication with two levels also provides performance comparable to sufferage and DFPLTF that assume complete knowledge about characteristics of jobs and resources.

2.2.4 RR Scheduling

The authors in [19] propose a round robin (RR) algorithm. The tasks are organized in a ring. At the beginning of scheduling, every processor is assigned one task. Whenever a processor finishes execution, a task, which is not yet assigned, is picked from the ring in a round robin fashion and allocated to the processor. This process continues until all tasks in the ring are assigned. After all tasks have been assigned, whenever a processor finishes executing the task, the completed task is removed from the ring and a replica of a task selected from the ring in a round robin fashion is allocated to the processor. Whenever a task or one of its replicas finishes execution, other replicas are killed and the task is removed from the ring. This process continues until all tasks finish execution. This algorithm also does not use any task or resource characteristic information to take a scheduling decision. The performance of RR scheduling was compared with DFPLTF, sufferage, min-min, max-min and workqueue scheduling. The results of experiments show that max-min provided the best performance followed by RR. Similar to the workqueue with replication, the disadvantage of RR scheduling is wastage of CPU cycles.

2.2.5 Commercially Available Scheduling Systems

This section describes the commercially available schedulers developed for a grid environment. Though called schedulers, they include the functionality of matching of jobs to resources.

2.2.5.1 AppLeS

The AppLeS (Application Level Scheduling [3]) parameter sweep template allows efficient deployment of parameter sweep applications over the grid [9]. Although parameter sweep applications are structured as a set of independent experiments, some experiments may share large input files and produce large output files. The AppLeS parameter sweep template focuses on the efficient co-location of data and experiments. It is critical for performance that hosts within a site can share data in local disks efficiently. It is an extension to the sufferage heuristic (described in Section 2.2.2.3): called the xsufferage heuristic. In xsufferage, the sufferage value is computed with cluster level MCTs.

2.2.5.1.1 XSufferage Vs Sufferage

The sufferage heuristic does not capture the file locality issues as expected. Consider a task T0 that requires a file stored on a remote cluster. If the cluster contains two machines with identical performance, then both the machines achieve nearly the same MCT for the task. If the file is of significant size as compared to the network bandwidth, it is possible that these two hosts lead to the best and second best MCT for the task. The sufferage value of the task will be zero, giving it low priority. Other tasks will be scheduled first, eventually forcing T0 to be scheduled somewhere else (on some other

cluster) which may require a file transfer. This impacts the overall application performance and makespan.

XSufferage modifies the definition of sufferage. For each task and each cluster, the task's MCT is computed for hosts in the given cluster and is called the cluster level MCT. The cluster level sufferage value is computed as the difference between the best and second best cluster-level MCTs. The task with the highest cluster-level sufferage is given the highest priority. It is scheduled on the host that achieves the lowest cluster level MCT.

2.2.5.2 Nimrod-G (Economic Grid Resource Broker)

The Nimrod-G [7] uses a computational economy driven architecture [1] for managing resources and scheduling task farming applications on large-scale distributed resources. It is a tool for execution of parameter sweep applications over global computational grids. The parametric experiments are expressed using a declarative parametric modeling language. The users create an experiment that contains an application specification. The experiment contains the requirements in terms of deadline and budget constraints and the scheduling policy to be implemented by the broker. Nimrod-G uses resource management and scheduling algorithms based on economic principles. Three types of adaptive algorithms have been incorporated in the Nimrod-G Broker: Time Minimization within time and budget constraints, Cost Minimization within time and budget constraints, and None Minimization within time and budget constraints [8]. Specifically, it supports user-defined deadline and budget constraints for

schedule optimizations. A detailed description of the Nimrod-G scheduler is available in [7].

2.2.5.3 Condor-G

Condor is a high throughput computing system developed at the University of Wisconsin-Madison. Condor uses a matchmaking architecture [33]. Matchmaking uses classified advertisements and allows entities to publish queries/requirements as attributes. Entities, which either provide or require a service, advertise their characteristics and requirements in classified advertisements. Then, a matchmaking service (matchmaker) finds a match that satisfies the requirements of both the service provider and the entity requiring service. Condor-G [18] combines software from the Condor [33] and Globus [14] to enable resource sharing in a grid environment.

Chapter 3: The Simulation Model

This chapter gives details of the experimental set-up, the simulation model and the simulation toolkit used. It presents the various proposed matching policies in detail. It also provides an overview of the system parameters, workload parameters, performance measures and test conditions.

3.1 System Architecture

As discussed in Chapter 1, computational grids [13] are sharing environments in which various geographically distributed resources are made available to groups of remote users. The resources are a part of a “virtual organization” [15], which may comprise of several individuals and institutions. In the system investigated in this thesis, the resources of a particular node in the grid are organized in a cluster. As described in Section 1.3.2, clusters at nodes may be interconnected to form grids. In the research in this thesis, a node is assumed to have one cluster. The clusters at different nodes experience communication delays while intercommunicating with one another. Each node has its own decentralized grid information service, which has information about resources belonging to the cluster in that node. There are various factors, which encourage the use of a decentralized information service over the centralized architecture [24]. As the grid environment is likely to scale to a large number of resources shared by a large number of participants (institutions and individuals), it is not feasible for a central authority to administer such a large collection of distributed resources. Each cluster has a traffic generator, which generates jobs following a poisson arrival process. The jobs are submitted to the broker of that cluster, responsible for discovering and matching the job

to a resource based on the matching algorithm. The simulation model and the simulated entities are described in detail in Section 3.3. The simulation toolkit used in building the simulation model is described next.

3.2 The GridSim Toolkit

The simulation in this research has been developed using GridSim [6]. GridSim is a toolkit for modeling and simulation of parallel and distributed systems such as clusters and grids. GridSim is a part of the Gridbus Project, jointly sponsored by the University of Melbourne (Australia), Sun Microsystems (USA), and Victorian Partnership for Advanced Computing (VPAC) (Melbourne, Australia).

GridSim supports the simulation of grid entities—resources, users, application tasks, and schedulers/brokers—and their characteristics using discrete events. Local sharing policies at the resources: time-shared or space-shared may be defined. Other resource characteristics like CPU speed, operating system, and cost to access a resource may be specified.

3.2.1 Architecture of GridSim

The multilayer architecture of GridSim as given in [6] is shown in Figure 3-1. The first layer comprises of a scalable Java interface and the run time machinery: Java Virtual Machine (JVM). The implementation of JVM is available for both single and multiprocessor systems. The second layer comprises of SimJava [21], which is a discrete-event infrastructure built using the interfaces provided by the first layer.

Figure 3-1: Architecture of GridSim

The GridSim Toolkit focuses on the third layer and simulates grid entities like resources, grid information services using the discrete-event services offered by the lower layer; SimJava. The fourth layer is concerned with the simulation of schedulers or resource brokers. The matching policies proposed in this thesis are implemented in the broker. The final layer focuses on application and resource modelling with different scenarios. The scenarios are implemented using the services provided by the lower two layers.

3.3 Simulation Model

As discussed in Section 3.2.1, describing the architecture of GridSim, the toolkit enables the simulation of entities like grid resources, information services etc. The entities interact using the discrete event infrastructure provided by SimJava. Figure 3-2

illustrates the simulation model consisting of n nodes, and each node comprising of one cluster. Full connectivity between all clusters is assumed.

Figure 3-2: The Simulation Model

The entities that have been implemented in our simulations are described next.

3.3.1 Source

Each cluster has a traffic generator that generates jobs following a poisson arrival process.

3.3.2 Broker

Each cluster has a broker that receives the jobs generated by the source of that cluster. The broker is responsible for discovery of resource and matches the job to a

resource based on the defined matching criterion. The dynamic matching policies proposed by this research are implemented in the broker.

3.3.3 Regional Grid Information Service

As discussed in Chapter 2, Section 2.1.1, there are many factors that favour a decentralised grid information service in comparison to a centralized information service. A decentralized architecture has been used in our research, where every node has a regional grid information service entity that contains information about resources belonging to the cluster in that node. It may be recalled that there is one cluster per node. Whenever a resource is added to a cluster, it has to register itself with the regional grid information service of that node. The broker, while attempting to discover a resource queries the regional information service entities of both the local and remote nodes. In this thesis it is assumed that the broker gets information about the regional information service entities from a central GIS and then sends queries to remote clusters. However, such an assumption will not affect the relative performance of the matching policies and the conclusions drawn in this research.

3.3.4 Resource

The resources implemented in our simulation are computing resources and execute the jobs that are submitted to them by various brokers.

3.3.5 Router

The entities in one node/cluster are connected to a router. The routers of different nodes are connected via high-speed optical links. The routers are responsible for any transmission and queuing delays.

3.3.6 Links

The links connect the routers of different nodes and also the entities in a node to the router in that node. The links are responsible for propagation delays.

These simulated entities respond to various events sent to them by other entities. The interaction between the entities is governed by the dynamic matching policy that is implemented. The various dynamic matching algorithms that have been implemented in this thesis and the system and workload parameters are described next.

3.4 Dynamic Matching Policies

The minimum completion time algorithm is a benchmark for all the implemented policies. Note that as discussed in Section 2.2.1.6, the performance of the MET algorithm was observed to be inferior to that of MCT. MCT provides good performance as it takes matching decisions based on complete knowledge of characteristics of jobs and resources. It takes into account the performance of the job on a particular resource as well as the loading of the resource. The performance of the matching policies introduced as a part of this research is compared to the minimum completion time algorithm. This section describes in detail the matching policies that are introduced in this research. A description of the implementation of the simulations for these policies is included. The way the algorithms have been implemented, and the interaction between the entities are also described in detail.

3.4.1 Minimum Completion Time

In this algorithm, the job is submitted to the resource that will give the minimum completion time for that job. The steps in the execution of the minimum completion time algorithm are described:

1. The job generated by the source is submitted to the broker of that cluster.
2. The broker sends a request to the regional grid information services of the local clusters and remote clusters to send a list of resources.
3. After receiving a list of resources from a grid information service of a cluster, the broker sends a request to the resources of that cluster to send the completion time of the job if the job were to be executed on that resource. Along with this request, the broker sends information about the length of the job expressed in Million Instructions (MI) to the resources.
4. The resources send the computed completion time back to the broker. The completion time of the job is calculated by each resource: as the sum of the start time of the job and the execution time of the job requesting a resource. The start time of the job is the time at which the jobs currently enqueued at the resource are completed.
5. The job is then submitted to the resource that gives the minimum completion time for the job.
6. If multiple jobs are submitted to a resource, first come first served scheduling policy is used.

3.4.2 Class Based

The minimum completion time algorithm takes matching decisions based on complete information about characteristics of jobs and resources. This may not be feasible in reality. The main contribution of this thesis is the introduction of a class-based algorithm. This algorithm does not need exact *a priori* information about the job and the resources. A class is associated with the job depending on its job length: SMALL, MEDIUM, and LARGE for example. Each resource in the grid also has a class associated with it. A job of a particular class gets allocated to a resource that is designated to perform jobs of the same class. As the jobs in one class cannot interfere with jobs in another class, a job in class “SMALL” cannot queue up behind a job in the “LARGE” class. However, this may be possible in MCT.

The class-based matching can improve the response time of shorter jobs if they happen to arrive in the system after a few large jobs have arrived or are executing in the system. The shorter jobs have to look for resources in their own class. Jobs of a particular class are submitted to a resource that has the same class as that of the job. For multiple jobs submitted to the same resource, first come first served scheduling policy at the resources is used.

Three types of class-based matching policies have been introduced in this research: class based, class based round robin, and class based with load information. These policies use different degrees of knowledge of the system and its state. The implementation of the class based matching policy is described in this section. The other two policies are described in the subsequent sections.

The steps in the execution of the class based (CB) matching algorithm are described below:

1. The job is submitted by the source of a cluster to the respective broker.
2. The broker queries the regional grid information service of the local cluster.
3. The regional grid information service of the local cluster sends a list of resources to the broker.
4. The broker sends a request to the resources in the local cluster to send their class.
5. If such a resource is found, the job is submitted; otherwise the broker proceeds with querying a remote cluster. In this policy, the request is propagated to the next cluster in order. For example, from cluster 0 the request will first be propagated to cluster 1. If the request is not satisfied by cluster 1, the request will be propagated to cluster 2 and so on.
6. The remote clusters are queried until a resource of the required class is discovered.
7. The job is submitted to the first resource found that has the same class as that of the job.

3.4.3 Class Based Round Robin

In the simple class based algorithm described in the previous section, if the requirement of a resource is not met by the local cluster, the broker always sends the request to the next cluster in order and so on. Also, the job is submitted to the first resource of the appropriate class discovered by the broker. However, this can cause overloading of resources in certain situations. For example, consider the case of three

clusters: cluster 0, cluster 1, and cluster 2. If cluster 0 has a single (single resource in a cluster based example) resource of class SMALL and cluster 1 and cluster 2 each have a resource of class LARGE, all jobs of class LARGE from cluster 0 will be submitted to the resource in cluster 1, that will be discovered first. There will be a load imbalance between the resources of class LARGE in cluster 1 and cluster 2.

In the class based round robin (CB-RR) algorithm, the clusters to which the request is propagated, if not met locally are chosen in a round robin fashion. For the scenario discussed in the previous paragraph, if one job of class LARGE from cluster 0 is submitted to the resource in cluster 1, the next job of class LARGE from cluster 0 will be submitted to the resource in cluster 2, thus balancing the load between resources of the same class in different clusters.

3.4.4 Class Based with Load Information

The class based with load information (CB-LI) matching policy uses the knowledge of system state. In this case, all clusters are polled and apart from checking the class of the resource, the queue length (number of jobs in queue waiting to be executed) is also checked. The job is submitted to the resource that has the same class as the job and also the shortest queue length. Though the communication overheads in this case will be the same as MCT as all clusters and resources are polled, the benefit that no information about characteristics of job and resources is required still remains.

The algorithms use different degrees of knowledge of the system and its state. Experiments are performed to compare the performances of these strategies.

3.5 Performance Measures

In order to assess the performance of the class-based matching policies introduced in this research with respect to the minimum completion time algorithm; mean response time of jobs has been used as a performance indicator.

3.5.1 Mean Response Time (R_t)

Mean Response Time is the average response time of the jobs submitted to the brokers. Response time of a job is the difference between the times; the job is submitted at the broker and the time when the job is returned back to the broker by the resource after execution. It is calculated by taking the sum of the response times of all jobs submitted to brokers of their respective clusters divided by the total number of submitted jobs.

3.6 Parameters

This section describes the system and workload parameters of interest.

3.6.1 System Parameters

The system parameters of interest are presented next.

N_c : Number of clusters in the grid

This is the number of clusters/nodes comprising the grid. As described in Section 3.1, each node comprises of one cluster. The number of clusters is equal to the number of nodes in the grid.

N_r : Number of resources in each cluster

In this research, each cluster is assumed to have the same number of resources denoted by N_r .

N_{ri} : Number of resources in a class

This refers to the number of resources of a particular class in a cluster.

 D_d : Delay due to discovery of resources at the information service

The process of resource discovery in a grid requires querying the regional information service entities of the clusters that contain information about the resources belonging to a particular cluster. In our simulation model, whenever a job request is submitted to a broker, the broker queries the regional grid information service entities of the local cluster and remote clusters to get information about the resources in a particular cluster. The delay due to discovery at a particular cluster is the time between the time when the request for list of resources is received by the grid information service entity of that cluster and the time when the list is sent by the grid information service entity to the requesting broker. We have used a fixed value of D_d in our research [23].

 D_t : Transmission delay

- Transmission delay (D_t) is the time required to transmit all the bits on to the link.
- The transmission rate/speed is the number of bits that can be transmitted on to the link in one second. It is expressed in bits/second.
- Transmission delay of a packet is the ratio between the packet size and the transmission speed. If a packet comprises of L bits and the transmission rate between two entities is R bits/second, transmission delay is L/R seconds.
- In our simulation, the nodes are connected via high-speed optical links. Optical carrier level 12 (OC-12), which is a circuit that transmits 622 megabits per second, has been considered for such a link. In our simulation, the transmission

rate at routers, which connect the nodes, is taken as 622 Mbps. The routers are responsible for transmission delays between nodes. The transmission rate at entities in a node has been taken as 100 Mbps.

D_p : Propagation delay

After a bit is pushed on the link, it has to move from the transmitting entity to the receiving entity. Propagation delay is the time required to propagate from the transmitting entity to the receiving entity. A typical value of propagation delay between the entities in the same cluster has been taken as 1 microsecond. Propagation delay between nodes has been fixed at 25 milliseconds. Similar values have been used by other researchers (see [35] for example).

3.6.2 Workload Parameters

This section describes the workload parameters of interest.

λ : Job Arrival Rate

Each node has a source that generates jobs at a rate λ . The jobs are submitted to the broker of that cluster. Arrival rate is the number of jobs generated per second. The total job arrival rate to the system is the sum of arrival rates of the sources of all clusters.

N_{classes} : Number of classes of jobs

Depending on their service demand (discussed next), jobs may belong to different classes. In most experiments, two classes of jobs are used: SMALL and LARGE. The effect of number of classes is studied in one of the experiments and three job classes: SMALL, MEDIUM and LARGE are used. The probability of job belonging to class SMALL, LARGE or MEDIUM is denoted by p_s , p_l , and p_m respectively.

M_{SD}: Mean Service Demand

The jobs of each class are characterized by a mean and a distribution. In our simulations, the length of jobs of a particular class follows a uniform distribution characterized by an upper bound U_i and a lower bound L_i , where i denotes the class of job (for example SMALL or LARGE). The mean job length is given by $(U_i + L_i)/2$. The length of jobs in million instructions (MI) of class SMALL follows a uniform distribution U_{SMALL} : $U(2500, 7500)$ with mean length 5000 MI. The length of jobs (in MI) of class LARGE follows a uniform distribution U_{LARGE} : $U(100000, 110000)$ with mean length 105000 MI.

The service demand of a particular class of jobs is the ratio of the mean job length of that class and the speed of the resources. The speed of computing resources (CPU_{speed}) has been assumed to be 100 MIPS. Thus, the mean execution time of jobs of class SMALL is 50 seconds and mean execution time of jobs of class LARGE is 1050 seconds.

The sum of service demands of different classes of jobs multiplied by their respective probability of occurrence gives the mean service demand of jobs. In our experiments with two classes of jobs, p_s is taken as 0.95 and p_l is taken as 0.05. Therefore, the overall mean service demand of jobs is $(0.95 \cdot 50 + 0.05 \cdot 1050) = 100$ seconds. Similar mean service demands have been used by other researchers (see [8] for example).

The utilization of jobs of a particular class is given by the product of arrival rate, probability of job being of a particular class and the mean execution time of jobs of that class. Therefore, the utilization ratio between the jobs in class SMALL and jobs in class LARGE based on default parameters (summarized in Table 3-1) is given as,

$$\lambda * 0.95 * 50 / \lambda * 0.05 * 1050 = 0.9.$$

COV: Coefficient of Variation of Job Demands

The coefficient of variation is the ratio of standard deviation and the mean. For two uniform distributions of jobs of the two classes, SMALL and LARGE with a probability of occurrence of SMALL being p_s , the mean and variance are given [11],

$$\text{Mean} = (p_s (L_{\text{SMALL}} + U_{\text{SMALL}}) + (1 - p_s)(L_{\text{LARGE}} + U_{\text{LARGE}})) / 2$$

$$\text{Variance} = (A - B + C) / 3$$

Where,

$$A = p_s (L_{\text{SMALL}}^2 + L_{\text{SMALL}} * U_{\text{SMALL}} + U_{\text{SMALL}}^2)$$

$$B = (3/4)(p_s (L_{\text{SMALL}} + U_{\text{SMALL}}) + (1 - p_s)(L_{\text{LARGE}} + U_{\text{LARGE}}))$$

$$C = (1 - p_s)(L_{\text{LARGE}}^2 + L_{\text{LARGE}} * U_{\text{LARGE}} + U_{\text{LARGE}}^2)$$

Based on the default values of workload parameters the default value of COV is 2.184.

In order to study the effect of a given parameter on performance, a factor at a time approach has been used in our simulation experiments. All the parameters are held at their default levels (see Table 3-1) and the parameter of interest is varied.

Table 3-1: Default Values of System & Workload Parameters

Parameter	Value
N_C	3
N_r	1
N_{classes}	2
U_{SMALL}	U (2500,7500)
U_{LARGE}	U (100000,110000)
p_s	0.95
p_l	0.05
$\text{CPU}_{\text{speed}}$	100 MIPS
M_{SD}	100 seconds
D_t between clusters	622.08 Mbps
D_t between entities in the same cluster	100 Mbps
D_p between entities in the same cluster	1 microsecond
D_p between clusters	25 milliseconds
D_d	0.3 seconds
Utilization ratio	0.9
COV	2.184

3.7 Verification of Simulation Experiments

Little's Law has been applied to ensure the correctness of simulation experiments.

3.7.1 Little's Law

The correctness of the experimental system was verified by applying Little's Law. There are two ways to get the number of jobs in the system. One way is that the average number of jobs in the system (N_0) equals the sum of residence times (response time) of all jobs divided by the experiment time. Experiment time is the time from the beginning of the experiment to the time when the last job leaves the system. The second way is to apply Little's Law. Little's law relates the mean number of jobs in the system with the mean time spent in the system as follows:

$$\text{Mean number in the system } (N_1) = \text{Arrival rate} * \text{Mean Response Time}$$

Both, the values of N_0 and N_1 are calculated and the difference is observed. The validation was done at a high arrival rate. The percentage differences for all the implemented matching algorithms were observed to be below one percent.

3.8 Accuracy of Measurement

This section describes in detail the testing conditions and the precautions observed to get accurate measurements.

3.8.1 Initial Bias

The initial bias for a simulation run is removed by discarding the observations in the transient phase. To determine the transient phase, the number of jobs is gradually increased and the mean response time of jobs is observed. In the steady state, there is not much variation in the mean response time of jobs as the number of jobs is increased. The value of number of jobs after which the mean response time stabilizes is regarded as the end of transient phase. The transient phase in each experiment is determined at a high

arrival rate. As simulations with higher arrival rate generally take longer to achieve a steady state, discarding the same number of jobs at lower arrival rates will also ensure the removal of the transient phase. In the steady state, completion of first n number of jobs submitted by the traffic source to the broker is considered for response time calculations. However the arrivals from the traffic source are not stopped until these jobs are completed, as the effective mean response time would be lowered if there were no arrivals in the system. The jobs in the steady state to be considered for mean response time calculations are flagged.

3.8.2 Test Duration

Multiple runs of the simulation were carried out and each simulation was run long enough in the steady state in order to achieve the desired accuracy: interval of $\pm 5\%$ of the mean at a confidence level of 95%. The method of independent replications is used in calculating the confidence interval. The desired accuracy was achieved except for a few cases at higher arrival rates. However, at those arrival rates, the difference between the performances of algorithms was substantial and even with a confidence interval between 5-10 %, it could be safely concluded as to which scheduling policy is performing better.

Chapter 4: Performance of Class Based Matching Algorithms

This chapter describes the different simulation experiments performed in detail. Effect of various system and workload parameters on the performance of the proposed matching policies is discussed.

4.1 Objectives

The objectives of this chapter are outlined:

- To answer “How does class-based matching, which does not require detailed *a priori* information about jobs and resources perform with respect to the minimum completion time algorithm that matches jobs to resources based on complete knowledge of characteristics of jobs and resources.
- To be able to answer “What factors will determine the number of resources to be associated with each class?”
- To get insights into the performance of various matching algorithms and study the effect of different system and workload parameters on the performance of the proposed matching algorithms.

In the previous chapter, the proposed class-based matching algorithms were described. In this chapter, we describe results of simulation experiments conducted to study the performance of the various class-based algorithms and the minimum completion time algorithm.

Simulation experiments are conducted for various system and workload parameters to study their effect on the performance of class-based matching. Experiments are conducted by varying the following:

- Number of clusters; N_c (Section 4.3)
- Number of resources in each cluster; N_r (Section 4.4)
- Configuration of resources in each cluster (Section 4.4.1.1)
- Number of classes of jobs; N_{classes} (Section 4.5)
- Coefficient of variation of job demands; COV (Section 4.6)
- Mean Service Demand of jobs; M_{SD} (Section 4.7)

4.2 Performance of the Class Based Algorithm

In the first experiment, two clusters each having a single resource is considered. The performance of class-based matching algorithms with more resources in each cluster is discussed in subsequent sections. As described in Section 3.1, each cluster has its own traffic source. Unless mentioned otherwise, the source in each cluster generates jobs of class SMALL with a probability p_s and jobs of class LARGE with a probability $(1-p_s)$. In the simulations described, the value of p_s is set to 0.95. The service demand of jobs in each class follows a uniform distribution. In order to concentrate on the performance of the algorithms alone, no delays due to discovery and communication are considered in this experiment.

The performance of the system is tested for three algorithms: local allocation (LA), class based matching, and minimum completion time. In all these

policies, once the job is matched to a resource, it is scheduled on the resource based on a first come first served scheduling policy. In case of local allocation, all the jobs of a cluster are submitted locally to the resource in the same cluster irrespective of the class of the job. In case of class based matching, as described in the previous chapter, a class is associated with each resource. In a two class environment, the resource in the first cluster is designated to perform jobs of class SMALL only and the resource in the second cluster is designated to perform jobs of class LARGE. All jobs of class LARGE in both clusters are submitted to the resource in the second cluster and all jobs of class SMALL in both clusters are submitted to the resource in the first cluster. In case of local allocation, if a job in class SMALL were to follow a job in class LARGE, it may be significantly delayed because they execute on the same resource. The rationale behind class based matching is to reserve a resource for each class of jobs. In this case, even if a job in class SMALL were to follow a job in class LARGE, it would not be delayed by the LARGE job, as it would execute on the resource designated to perform jobs of class SMALL only. The job in class LARGE will be sent to the other cluster.

The arrival rate is varied. The other workload parameters are held at their default values as listed in Table 3-1. The relative performance of the matching policies is shown in Figure 4-1. The Class Based matching algorithm outperforms both LA and MCT algorithms. In case of both local allocation and MCT, a large job may be in the front of a small job. However in case of class-based matching, a job in class SMALL cannot be delayed by a job in class LARGE as they contend for resources designated to perform jobs of their class.

Figure 4-1 : Performance of class based matching for $N_c=2$

4.3 Effect of Number of Clusters

To study the effect of number of clusters on performance, the number of clusters is varied.

4.3.1 Performance of Class-Based Matching in Three Clusters

The first experiment performed is with the number of clusters equal to three. All other workload parameters are kept the same as the parameters in the earlier experiment with two clusters. It may be recalled from Section 3.6.2 that the default utilization ratio (ratio between the utilization of jobs in class SMALL and the utilization of jobs in class LARGE) is 0.9. Therefore, two resources (one in cluster 1 and one in cluster 2) are allocated to execute the jobs of class LARGE and one resource is allocated to execute jobs of class SMALL.

As shown in Figure 4-2, class based matching performed worse than MCT. This is attributed to the saturation of the resource allocated to execute jobs of class SMALL. The ratio of utilization of jobs of class SMALL to utilization of jobs of class LARGE is 0.9. However, the ratio of resources allocated to jobs of class SMALL to the resources allocated to jobs of class LARGE is 0.5. Therefore, the resource allocated to execute jobs of class SMALL saturates much earlier than the resources allocated to execute jobs of class LARGE. However, such an imbalance would decrease with the increase in the size of the grid. With grids involving a large number of organizations with a large number of resources, the load can be balanced more equally and this kind of situation is unlikely to arise.

In case of CB matching, if the requirement of a resource cannot be met locally, the request is propagated to the next cluster and so on. Whenever a resource of the required class is found, the job is submitted. Hence the jobs of class LARGE from cluster 0 would always be submitted to the resource in cluster 1. Moreover, the jobs in class LARGE generated at cluster 1 and cluster 2 will be executed locally. This causes an imbalance in load distribution between the resources in cluster 1 and cluster 2.

If a requirement of a resource cannot be met locally, the clusters should be queried in a round robin fashion in order to achieve a more uniform load distribution. This will ensure that all resources of a particular class are more equally utilized.

The relative performance of class based and class based round robin matching is shown in Figure 4-2. It can be observed that CB-RR matching gives us better values of response times than CB matching as the jobs from cluster 0 were submitted to the resource in cluster 1 and cluster 2 alternately. The CB-LI matching policy, which

matches a job to a resource based on the class of the job and also the load on the resources of the same class performs slightly better than CB-RR matching. However, the saturation arrival rate remained the same, as the cause of saturation of the system was the resource in cluster 0-executing jobs of class SMALL. MCT outperforms CB, CB-RR, and also the CB-LI matching policy. The class-based matching algorithms saturate over the arrival rate of 0.02. MCT seems to saturate at a higher arrival rate.

Figure 4-2: Mean Response Time of jobs for $N_c=3$

4.3.2 Performance of Class-Based Matching in Four Clusters

All other system and workload parameters remaining same, the number of clusters is increased to four. Thus there are a total of four resources, one in each cluster. As discussed in Section 3.6.2, the default utilization ratio between jobs of class SMALL and jobs of class LARGE is 0.9. Therefore, two resources are allocated to execute jobs of class SMALL and two resources are allocated to execute jobs of class LARGE.

In this case, it was expected that there would not be load imbalance as observed in the case of three clusters and the performance of class-based matching would be better than MCT. At low and medium load (job arrival rates less than or equal to 0.035 jobs/sec), CB-RR performed better than MCT. However, at higher load, CB-RR matching policy saturated earlier than MCT. On analysis it was observed that the resources of the same class were not equally utilized and some resources were more utilized than others. Even in case of CB-RR matching, more jobs of one class could be allocated to one resource than the other. This depends on the number of times the request for a resource of a particular class is propagated to a particular remote cluster if not satisfied locally. For example, consider clusters 0 and 1 each having a single resource designated to perform jobs of class SMALL and clusters 2 and 3 each having a single resource designated to perform jobs of class LARGE. In order to check the number of jobs of class LARGE generated by the source in cluster 0 that would be submitted to resources in cluster 2 and cluster 3 respectively, consider the first three jobs of class LARGE that arrive at cluster 0. For the first job of class LARGE, the request will be propagated to cluster 1 (the next cluster to be checked is 1) and as it cannot be satisfied by the resource in cluster 1, it will be propagated to cluster 2 and will be satisfied by the resource in cluster 2. For the second job of class LARGE, the request will be propagated to cluster 2 (the next cluster to be checked is 2) and will be satisfied by the resource in cluster 2. The third request will be propagated to cluster 3 (the next cluster to be checked is 3) and will be satisfied by the resource in cluster 3. Similarly, considering the first three jobs of class LARGE in cluster 1, two jobs will get allocated to the resource in cluster 2 and one job will get allocated to the resource in cluster 3. The resource in cluster

2 gets a higher share of jobs from cluster 0 and cluster 1 than the resource in cluster 3, thereby explaining the uneven load distribution. The performance of CB-RR matching depends on how resources in different clusters are mapped to the different classes.

To ensure load balancing, the performance of class based with load information algorithm was tested. In this case, all clusters are checked to find resources of the required class and the job is submitted to the resource with the shortest job queue. The class based with load information algorithm outperformed MCT.

The relative performance of the matching policies is shown in Figure 4-3 and Figure 4-4.

Figure 4-3: Mean Response Time of jobs at all loads for $N_c=4$

The CB-LI algorithm gives the best performance at medium and high loads (job arrival rates higher than or equal to 0.025 jobs/sec). Even at low load (job arrival rate of 0.01 jobs/sec), the performance of CB-LI matching is comparable to MCT. The

performance of CB-RR matching is better than MCT at medium load. However, at high load, CB-RR saturates earlier than MCT for the reason explained earlier in this section.

Figure 4-4: Mean Response Time of jobs at low & medium load for $N_c=4$

4.3.3 Performance of Class-Based Matching in Six Clusters

All other system and workload parameters held at their default values (see Table 3-1); the number of clusters is increased to six. There are a total of six resources, with one resource in each cluster. It may be recalled that the default ratio of utilization of jobs of class SMALL to the utilization of jobs of class LARGE is 0.9. Therefore, three resources are allocated to execute jobs of class SMALL and three resources are allocated to execute jobs of class LARGE. Since class based round robin matching demonstrated an inferior performance in the previous section, only the performance of MCT and class based with load information algorithm is studied here. The mean response times obtained with these policies are shown in Figure 4-5 and Figure 4-6.

Figure 4-5: Mean Response Time of jobs at all loads for $N_c=6$

Figure 4-6: Mean Response Time of jobs at low and medium load for $N_c=6$

The class based with load information algorithm provides comparable performance to MCT at low load (job arrival rate of 0.01 jobs/second). At high loads (job

arrival rates higher than or equal to 0.03 jobs/sec), CB-LI matching outperforms MCT. At low loads, MCT provides good load balancing as well. This is because the jobs are matched to resources that would give the minimum completion time for the job. At high loads, MCT saturates earlier than CB-LI matching. This can be attributed to the fact that MCT matches the current job optimally without taking care of the future workload. There could be many small jobs queued after long jobs at high load. Such a situation does not arise in CB-LI matching as the jobs are always queued up at a resource designated to perform jobs of that class. Thus the situation in which a short job gets queued up behind a long job will not arise. The load balancing approach of submitting the job at a resource with the smallest queue length attempts to balance the load amongst resources of the same class.

4.4 Effect of Number of Resources in Each Cluster

To study the effect of number of resources in each cluster, all other system and workload parameters are kept the same as the default parameters (see Table 3-1) and the number of resources in each cluster is varied from one to two to four.

4.4.1 Performance with Two Resources in Each Cluster

When there are multiple resources in a cluster there could be multiple ways in which resources in different clusters may be mapped to classes. The performance of class based matching may become dependent on the system configuration (number of resources of a particular class in a particular cluster). For example, consider the case of two resources in each cluster and number of clusters equal to three. There are a total of six resources. Based on the default utilization ratio between jobs of class SMALL and

jobs of class LARGE, three resources are to be allocated to jobs of class SMALL and three resources are to be allocated to jobs of class LARGE. Two configurations of placement of resources within clusters are possible.

- Configuration one: Each cluster with one resource allocated to jobs of class SMALL and one resource allocated to jobs of class LARGE.
- Configuration two: Two resources of class SMALL in one cluster, two resources of class LARGE in the second cluster and a resource of class SMALL and a resource of class LARGE in the third cluster.

The question arises as to how the performance is related to the mapping of resources to classes of jobs. This question is studied in the next sub-section.

4.4.1.1 Effect of Configuration of Resources in Each Cluster

The effect of configuration of resources in a cluster was studied for the case with two resources in each cluster. The default values of system and workload parameters are used (see Table 3-1); only the number of resources in each cluster is changed to two. With three clusters, there are a total of six resources. As discussed in the previous section, in order to have the ratio of resources allocated to the two classes match up closely with the ratio of job utilization in classes, three resources are allocated to jobs of class SMALL and three resources are allocated to jobs of class LARGE. The two possible configurations of placement of resources within clusters as described earlier are discussed in the following sub sections.

4.4.1.1.1 Configuration One

In this case, each cluster has a resource allocated to execute jobs of class SMALL and a resource allocated to execute jobs of class LARGE. The values of mean response times obtained with the different matching policies are shown in Figure 4-7.

Figure 4-7: Mean Response Time of jobs in configuration one for $N_r=2$

It may be recalled that in case of CB-RR matching, the class of the resource in local cluster is checked first before propagating the request to remote clusters. Therefore, all jobs in class SMALL and LARGE execute locally as all the clusters have a resource allocated to execute jobs of class SMALL and jobs of class LARGE. CB-LI matching takes into account the load at various resources while making a matching decision, and distributes the load more evenly across resources as compared to CB-RR matching. CB-RR provides better performance than MCT at high load (job arrival rates higher than 0.05 jobs/sec) whereas CB-LI outperforms MCT at both medium and high loads (job arrival

rates higher than or equal to 0.04 jobs/sec). Even at a low arrival rate of 0.01 jobs/sec, there is a marginal difference in the performance of MCT and CB-LI matching.

4.4.1.1.2 Configuration Two

In this case, cluster 0 has two resources of class SMALL, cluster 1 has one resource of class SMALL and one resource of class LARGE, and cluster 2 has two resources of class LARGE. The values of mean response time obtained with the different matching policies are shown in Figure 4-8.

Figure 4-8: Mean Response Time of jobs in configuration two for $N_r=2$

The performance of class based round robin and class based with load information algorithms is compared with the performance of MCT. CB-RR saturates earlier than MCT. This can be attributed to the load imbalance between resources of the same class in a cluster. Though, class based round robin matching attempts to provide load balancing at cluster level by polling the clusters in a round robin fashion if the request is not satisfied

locally, the job is submitted to the resource of the same class that sends the reply to the broker first. The class based with load information matching provides the best performance as it provides load balancing both at cluster level and at the resource level. With this algorithm, all resources are polled and the job is submitted to the resource that has the shortest job queue amongst the resources of the required class.

4.4.2 Performance with Four Resources in Each Cluster

In this case, all other system and workload parameters remaining the same as the default parameters (see Table 3-1), the number of resources in a cluster is fixed at four. There are a total of twelve resources (four in each cluster). Six resources are allocated for jobs of class SMALL (two in each cluster) and six resources are allocated for jobs of class LARGE (two in each cluster). This is based on the default value of utilization ratio (0.9) between jobs of class SMALL and jobs of class LARGE. For the case with two resources in each cluster, it was observed that the class based with load information approach performed the best irrespective of the configuration of resources in a cluster as it provided good load balancing amongst resources of the same class in a cluster. Therefore, only the performance of MCT and class based with load information approach is compared here.

The relative performance of the implemented policies is shown in Figure 4-9 and Figure 4-10. MCT provides good load balancing and better performance at low loads (job arrival rates less than 0.1 jobs/sec). However at medium and higher loads, similar to other cases discussed in the previous sections MCT saturates much earlier than CB-LI matching. In case of MCT, at high loads, there could be many small jobs queued behind

long jobs leading to high mean response times and earlier saturation. Such a situation does not arise in case of class-based matching as the jobs contend for resources in the same class.

Figure 4-9: Mean Response Time of jobs at all loads for $N_r=4$

Figure 4-10: Mean Response Time of jobs at low & medium load for $N_r=4$

4.5 Effect of Number of Classes of Jobs

The effect of number of classes of jobs is studied for workloads with same mean service demand and coefficient of variation. The number of classes is varied from two to three. The default values of mean service demand and coefficient of variation (from Table 3-1) have been used. The effect of number of classes is studied for the case with four resources in each cluster and with three clusters in the system. The number of classes will affect the number of resources allocated for each class of jobs. The default case of having a single resource in each cluster is not chosen, as it would have become mandatory to allocate one resource for each class. Effect of number of classes can be better studied with a larger number of resources in each cluster. Hence, the scenario of four resources in each cluster was selected.

The summary of system and workload parameters for the case with two classes and three classes is given in Table 4-1.

The values of mean response times obtained for the case with three classes are shown in Figure 4-11 and Figure 4-12. The performance with two classes has already been shown in Section 4.4.2. In both the cases for two classes and three classes of jobs, MCT gives a slightly better performance at low loads but saturates earlier than CB-LI matching. However, the difference between the performance of MCT and CB-LI matching at low loads (job arrival rates less than 0.1 jobs/sec) is greater in the case with three classes.

As discussed in the earlier sections, in class-based matching, the number of resources allocated to a particular class should be proportional to the utilization of jobs of that class. As described in Section 3.6.1, the utilization of jobs of a particular class is a

product of the arrival rate, probability of the job being of that class and the service demand of jobs of that class.

For the case of two classes the utilizations of jobs are,

Utilization of jobs of class SMALL = $\lambda * 0.95 * 50$

Utilization of jobs of class LARGE = $\lambda * 0.05 * 1050$

Thus the ratio of utilization of jobs of class SMALL to the utilization of jobs of class LARGE is 0.9. There are a total of twelve resources (four in each cluster), out of which six are allocated to jobs of class SMALL and six resources are allocated to jobs of class LARGE.

Table 4-1: System & Workload Parameters to study the effect of number of classes on Mean Response Time

Parameter	Two Classes	Three Classes
N_C	3	3
N_{classes}	2: SMALL & LARGE	3: SMALL, MEDIUM & LARGE
N_r	4	4
Service Demand of jobs of class SMALL	U (25,75)	U (25,75)
Service Demand of jobs of class MEDIUM	Not Applicable	U (1000,1050)
Service Demand of jobs of class LARGE	U (1000,1100)	U (1050,1100)
p_s	0.95	0.95
p_m	Not Applicable	0.025
p_l	0.05	0.025

Figure 4-11: Mean Response Time of jobs at all loads for $N_{\text{classes}}=3$

Figure 4-12: Mean Response Time of jobs at low & medium load for $N_{\text{classes}}=3$

In the case of three classes, the utilizations of jobs of the three classes are,

Utilization of jobs of class SMALL = $\lambda * 0.95 * 50$

Utilization of jobs of class MEDIUM= $\lambda*0.025*1025$

Utilization of jobs of class LARGE = $\lambda*0.025*1075$

The ratio of utilization of jobs of class SMALL and MEDIUM with respect to jobs of class LARGE is 1.76 and 0.95 respectively. Three resources are allocated to jobs of class LARGE, three to jobs of class MEDIUM and six resources to jobs of class SMALL. The ratio of the number of resources allocated to jobs of class SMALL and MEDIUM with respect to the number of resources allocated to jobs of class LARGE is two and one respectively. In the case of two classes, the percentage difference between the ratio of utilization of jobs of the two classes and the ratio of resources allocated to the two classes is 10% (the ratio of utilization of jobs in the two classes is 0.9 and the ratio of resources allocated to the jobs in the two classes is 1). In the case of three classes, the percentage difference between ratio of utilization of jobs and ratio of resources allocated to jobs of class SMALL and MEDIUM with respect to jobs of class LARGE is 12% (ratio of utilization of jobs in class SMALL and LARGE is 1.76 and the ratio of resources allocated to the jobs of these two classes is 2) and 5% (ratio of utilization of jobs in class MEDIUM and LARGE is 0.95 and the ratio of resources allocated to the jobs of these two classes is 1) respectively. This affects the balancing of loads amongst the resources allocated to the different classes. The performance of class-based matching is determined by how closely the number of resources allocated to a particular class matches the utilization of jobs of that class.

4.6 Effect of Coefficient of Variation of Job Demands

The effect of coefficient of variation of job demands on the performance of class-based matching is studied. The performance is studied in three clusters, with one resource in each cluster. The number of classes is two (default): SMALL and LARGE. The performance is studied for values of coefficient of variation 1.09, 1.5, 2.94, and 3.41. These values of COV were derived from the equations of mean and variance given in Section 3.6.2. Given M_{SD} and COV, there are two equations and five unknowns. As a result there was no direct way of solving for L_{SMALL} , U_{SMALL} , L_{LARGE} and U_{LARGE} . A trial and error method was therefore used. By fixing p_s at 0.95, various values of these four parameters were chosen until a value close to the desired COV was obtained.

The workload parameters for these values of coefficient of variation of job demands are given in Table 4-2. The system parameters are the same as default parameters (see Table 3-1). With three clusters and a single resource in each cluster, for each value of coefficient of variation, the resources are mapped to classes based on the ratio of utilization of jobs in class SMALL and jobs in class LARGE for that value of coefficient of variation.

The performance of CB-RR and CB-LI matching is compared with that of MCT. The relative performance of the different policies for coefficient of variation of job demands 1.09, 1.5, 2.94, and 3.41 is shown in Figure 4-13, Figure 4-14, Figure 4-15, and Figure 4-16 respectively.

Table 4-2: Workload Parameters for various coefficient of variation

Coefficient of Variation of Job Demands	Service Demand of jobs of class SMALL	Service Demand of jobs of class LARGE	Probability of occurrence of jobs of class SMALL (p_s)	Probability of occurrence of jobs of class LARGE ($1-p_s$)
1.09	U (50,100)	U (550,600)	0.95	0.05
1.5	U (20,120)	U (120,1220)	0.95	0.05
2.94	U (25,40)	U (1365,1400)	0.95	0.05
3.41	U (30,35)	U (35,2730)	0.95	0.05

For a COV of 1.09, two resources are allocated to jobs of class SMALL and one resource is allocated to jobs of class LARGE. As can be seen in Figure 4-13, MCT performs better than CB-RR at all arrival rates. CB-LI gives performance comparable to MCT at low and medium load (job arrival rates less than or equal to 0.025 jobs/sec). At high load (job arrival rates higher than or equal to 0.0275), CB-LI saturates earlier than MCT.

For a COV of 1.5, two resources are allocated to jobs of class SMALL and one resource is allocated to jobs of class LARGE. As can be seen from Figure 4-14, CB-LI outperforms both MCT and CB-RR. MCT performs better than CB-RR matching at low and medium load (job arrival rates less than or equal to 0.025 jobs/sec), but at high loads (job arrival rates higher than or equal to 0.0275 jobs/sec) MCT saturates earlier than CB-RR.

Figure 4-13: Mean Response Time of jobs for C.O.V=1.09

Figure 4-14: Mean Response Time of jobs for C.O.V=1.5

Figure 4-15: Mean Response Time of jobs for C.O.V=2.94

For COV of 2.94 and 3.41, one resource is allocated to jobs of class SMALL and two resources are allocated to jobs of class LARGE. The CB-RR and CB-LI matching algorithms perform significantly better than MCT at a COV of 2.94 and 3.41 for all arrival rates (see Figure 4-15 and Figure 4-16).

The higher the coefficient of variation of job demands, the better is the performance of class-based matching. With higher coefficient of variation of job demands, there will be large differences between jobs of class SMALL and jobs of class LARGE. Since the overall mean is fixed, the jobs of class SMALL will be smaller and jobs of class LARGE will be larger. Thus, for higher coefficient of variation of job demands, if a job of class SMALL were to queue up behind a job of class LARGE as in case of MCT, the delay is expected to be greater than in case of a workload with low coefficient of variation of job demands.

Figure 4-16: Mean Response Time of jobs for C.O.V = 3.41

The relative performance of the different algorithms for the different values of coefficient of variation of job demands at an arrival rate of 0.02 jobs/second and 0.0275 jobs/second is shown in Figure 4-17 and Figure 4-18 respectively.

The performance of CB-RR and CB-LI matching improves as the coefficient of variation of job demands increases.

As can be seen from Figure 4-17 and Figure 4-18,

- CB-RR performs worse than MCT for a COV of 1.09 at both the arrival rates. CB-LI provides comparable performance to MCT at the lower arrival rate; however at the higher arrival rate MCT outperforms CB-LI.
- For COV of 1.5, CB-LI outperforms both MCT and CB-RR. MCT performs better than CB-RR at the lower arrival rate; however CB-RR outperforms MCT at the higher arrival rate.
- For COV of 2.94 and 3.41, both CB-RR and CB-LI outperform MCT.

This indicates that the performance of class-based matching algorithms improves as the variability in job demand increases.

Figure 4-17: Performance of various policies for different values of COV at $\lambda=0.02$

Figure 4-18: Performance of various policies for different values of COV at $\lambda=0.0275$

4.7 Effect of Mean Service Demand of Jobs

The effect of M_{SD} on the relative performances of MCT and class-based matching algorithms is investigated. The system parameters are the default system parameters with number of clusters equal to three and there is one resource in each cluster. The mean service demand of jobs of class SMALL and jobs of class LARGE is doubled with respect to the default values (given in Table 3-1). The distribution of service demand of jobs of class SMALL is modified from $U(25, 75)$ to $U(50, 150)$ and the distribution of service demand of jobs of class LARGE is modified from $U(1000, 1100)$ to $U(2000, 2200)$. The values of mean response time obtained with MCT, CB-RR, and CB-LI matching and the relative performance of the policies is shown in Figure 4-19.

Figure 4-19: Effect of M_{SD} on Mean Response Time

As the mean service demand of the jobs is doubled, the system saturates much earlier as compared to the saturation at the default workload parameters. The utilization of jobs is directly proportional to the mean service demand of the job, thus increasing the

utilization of resources. However, the relative performance of CB-RR and CB-LI matching with respect to MCT remains the same (see Figure 4-2 and Figure 4-19).

Similar relative performances of CB-RR and CB-LI matching with respect to MCT are expected for other values of M_{SD} . Therefore, experiments are not performed for any other value of mean service demand of jobs.

4.8 Effect of Communication Delays

For CB and CB-RR matching policies, it may be recalled that all clusters may not be polled, and the job is submitted to the first resource discovered in that class. The communication overheads of these policies will be less than that of MCT, as in case of MCT, all clusters and resources are polled before making a matching decision.

The communication overheads of CB-LI matching are similar to that of MCT, as in order to check the load on the resources, resources in all clusters are polled. The result of the experiments discussed in the previous sections show that despite of the same communication overheads as that of MCT, and unavailability of detailed information about characteristics of jobs, CB-LI gives performance comparable to that of MCT for some cases and outperforms MCT in some other cases.

4.9 Class Based Full Round Robin Algorithm

Another class-based matching policy is analysed here. For the CB-RR matching algorithm discussed in the previous sections, the broker first queries the information service entity of the local cluster and checks the resources in the local cluster to find a resource of the required class. The request is propagated to the remote clusters in a round robin fashion only if it cannot be satisfied by a resource in the local cluster. In class based

full round robin (CB-FRR) matching, the clusters are polled in a round robin fashion including the local cluster irrespective of the availability of resources in the local cluster. The policy is proposed to provide a more even distribution of load as compared to class based round robin matching.

The performance of CB-FRR has been tested for the case with three clusters and two resources in each cluster. Default values of workload parameters are used. Based on the default utilization ratio of 0.9, three resources are allocated to jobs in class SMALL and three resources are allocated to jobs in class LARGE.

It may be recalled from Section 4.4.1 that with two resources in each cluster, there are two ways in which the resources in the three clusters may be mapped to the two classes: SMALL and LARGE.

In configuration one, each cluster has a resource with class SMALL and a resource with class LARGE. The performance of CB-FRR matching with respect to MCT, CB-RR, and CB-LI in configuration one is shown in Fig 4-20.

Figure 4-20: Performance of CB-FRR in configuration one for $N_r = 2$

CB-LI outperforms all other policies. CB-FRR provides better load balancing than CB-RR matching.

In configuration two, cluster 0 has two resources of class SMALL, cluster 1 has one resource of class SMALL and one resource of class LARGE, and cluster 2 has two resources of class LARGE.

The performance of CB-FRR matching with respect to MCT, CB-RR, and CB-LI in configuration two is shown in Figure 4-21.

Figure 4-21: Performance of CB-FRR in configuration two for $N_r = 2$

MCT outperforms all the other policies at low and medium load (job arrival rates less than 0.04 jobs/sec). However, at higher load, CB-LI outperforms MCT. Both CB-RR and CB-FRR get saturated much earlier than MCT. This can be attributed to the fact that these policies attempt to balance the load at cluster level but do not balance the load amongst resources of the same class in a cluster.

The results presented in this thesis do not indicate any strong advantage of CB-FRR. Another approach to balance the load amongst resources of the same class in a cluster is to poll the clusters in a round robin fashion and to allocate jobs to resources of the same class in a cluster also in a robin fashion. This approach would also attempt to distribute the load both at the cluster and at the resource level. Investigation of such an algorithm forms an interesting direction for future research.

The conclusions drawn from the experiments in this chapter are summarised in the next chapter.

Chapter 5: Conclusions

This chapter provides a summary of this research and discusses the directions for future research.

5.1 Summary

A popular algorithm described in the literature is minimum completion time that requires exact *a priori* knowledge of job and system characteristics. It is not always possible to have a complete *a priori* knowledge about the characteristics of a job while making a matching decision. Class-based matching algorithms introduced in this research do not require detailed knowledge of the job characteristics in order to match a job to a resource. An important observation made in this thesis is that class-based matching displays performance comparable or superior to the minimum completion time algorithm under a broad range of workload conditions, when the variability in job demand is high.

In the class-based matching approach, a class is associated with a job. In this research, the classification of jobs into classes is based on distribution of service demand of jobs. The resources participating in the grid also have a class associated with them. The jobs of a particular class are executed on a resource of the same class only.

The performance of different types of class-based matching algorithms is observed under different configurations by varying system and workload parameters. The performance of the class-based matching algorithms is compared with the minimum completion time algorithm that takes matching decisions based on complete knowledge of job characteristics. With MCT, all resources in the grid are checked and the job is

submitted to the resource that would provide the minimum completion time for the job. Various class-based matching algorithms that use different degrees of knowledge of the system and its state are introduced in this research. The performance of these class-based strategies is evaluated by running a series of simulation experiments. Experiments are performed by varying the number of clusters, number of resources in a cluster, number of classes of jobs, job arrival rate, as well as the coefficient of variation and mean service demand of jobs.

The main benefit of class-based matching with respect to MCT is that it does not require detailed information about characteristics of jobs before making a matching decision. In a real system, where the jobs may be continuously arriving, it may be difficult to implement MCT that matches jobs to resources based on complete information about characteristics of jobs. The performance of class-based matching depends on how well the resources can be divided amongst classes.

Different types of class-based matching policies performed differently with different set of system and workload parameters. The performance results are described in the thesis. The effects of important parameters on performance of class based (CB), class based RR (CB-RR), and class based with load information (CB-LI) matching policies are summarised here.

- For less than three clusters with a single resource in each cluster, simple class based matching is observed to demonstrate good performance. At first, the local cluster is checked for the resource of the required class. The job is submitted locally if a resource is found, otherwise the request is propagated to the remote cluster.

- For number of clusters equal to three and single resource in each cluster, CB-RR provides more even distribution of load on resources. In case of CB-RR matching, if the request cannot be met locally, the request is propagated to remote clusters in a round robin fashion.
- For more than three clusters with a single resource in each cluster, CB-RR may not equally utilize the resources of the same class. This depends on how the resources in different clusters are mapped to the classes. For clusters with multiple resources of the required class, the load may be unevenly distributed on the resources as the job is submitted to the resource that first sends the reply to the broker.
- In order to provide load balancing both at cluster and resource level (the two scenarios discussed earlier), class based with load information matching is introduced. In this case all clusters and resources are checked and the job is submitted to the resource with the shortest job queue amongst resources of the required class.
- If the resources are appropriately allocated to different classes, class-based matching demonstrates a superior performance in comparison to MCT at higher loads. In MCT, there can be queuing of a large number of small jobs behind large jobs at higher arrival rates. Such a situation does not arise in class-based matching as the jobs are classified based on their service demand and jobs of different classes do not contend for the same resource.
- The relative performance of class-based algorithms with respect to MCT

improves with increase in coefficient of variation of job service demands.

5.2 Future Research

A number of issues that warrant investigations are presented in this section.

- The performance of class-based matching is governed by the utilization of jobs of a particular class. The system saturates when a resource executing jobs of a particular class saturates. It may be possible that a resource executing jobs of a particular class may be saturated, while the resource executing jobs of the other class may be poorly utilized. An adaptive strategy is proposed as a part of future research. In the event of high utilization of resources designated to perform jobs of a particular class, the jobs of that class should be submitted to resources with lesser utilization, that are otherwise allocated to perform jobs of another class. It will be interesting to compare the performance of such an adaptive strategy with the strategies discussed in this thesis.
- Most of the work in this research was based on the workload comprising of a set of independent sequential jobs. Investigation of the proposed policies for other workloads that contain parallel jobs needs investigation.
- In the current research, if a job is submitted to a resource, it runs to completion. Migration strategies can be considered to migrate jobs from more loaded resources in one class to comparatively lightly loaded resources in another class to further improve system performance.
- The performances of the different class-based matching algorithms introduced in this research have been examined for a synthetic workload. Testing the

performance of the proposed algorithms with real life workload traces provides an interesting dimension for future work.

- The performances of the proposed algorithms should be tested with an actual computing grid in which the incoming jobs are matched to resources based on the matching policies introduced in this research.
- The effectiveness of the class-based matching strategies needs to be investigated for various network topologies and communication delays.
- Research presented in this thesis has focused only on computing resources. Investigation of a system in which multiple resources are to be co-allocated to jobs forms an interesting direction for future research.

REFERENCES

- [1] Abramson D., Buyya R. and Giddy J., "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker," *Journal of Future Generation Computer Systems*, Volume 18, Issue 8, October 2002, pp. 1061-1074.
- [2] Allcock B., Bester J., Bresnahan J., Chervenak A.L., Foster I., Kesselman C., Meder S., Nefedova V., Quesnel D. and Tuecke S., "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," in *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies (MSS'01)*, Washington, DC, April 2001, pp.13-28.
- [3] Berman F., Wolski R., Figueira S., Schopf J.M., and Shao G., "Application-Level Scheduling on Distributed Heterogeneous Networks," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, Pittsburgh, PA, November 1996, pp. 39.
- [4] Bester J., Foster I., Kesselman C., Tedesco J. and Tuecke S., "GASS: A Data Movement and Access Service for Wide Area Computing Systems," in *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems (IOPADS-99)*, New York, NY, May 1999, pp. 78-88.
- [5] Braun T.D., Siegel H.J., Beck N., Boloni L., Maheswaran M., Rerther A.I., Robertson J.P., Theys M.D. and Yao B., "Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," in *Proceedings of the Symposium on Reliable Distributed Systems (SRDS 1998)*, West Lafayette, IN, October 1998, pp. 330-335.
- [6] Buyya R. and Murshed M., "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Journal of Concurrency and Computation: Practice and Experience (CCPE)*, Volume 14, Issue 13-15, Wiley Press, December, 2002, pp.1175-1220.
- [7] Buyya R., Abramson D., and Giddy J., "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid," in *Proceedings of the 4th International Conference on High Performance Computing in the Asia Pacific Region, (HPC ASIA'2000)*, Volume 1, Beijing, China, May 2000, pp. 283-289.
- [8] Buyya R., Abramson D., and Giddy J., "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications," in *Proceedings of the 2nd Workshop on Active Middleware Services (AMS 2000) held in conjunction with 9th IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, Pittsburgh, PA, August 2000.

- [9] Casanova H., Obertelli G., Bermand F., and Wolski R., "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Proceedings of the Super Computing Conference (SC'2000)*, Dallas, TX, November 2000, pp 75-76.
- [10] Czajkowski K., Fitzgerald S., Foster I., and Kesselman C., "Grid Information Services for Distributed Resource Sharing," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, August 2001, pp. 181-194.
- [11] Czajkowski K., Foster I., Karonis N., Kesselman C., Martin S., Smith W., and Tuecke S., "A Resource Management Architecture for Metacomputing Systems," in *Proceedings of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, Orlando, FL, March 1998, pp. 62-82.
- [12] Fernández B.D., "Allocating Modules to Processors in a Distributed System," *IEEE Transactions on Software Engineering*, Volume15, Issue11, November 1989, pp.1427-1436.
- [13] Foster I. and Kesselman C., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc, San Francisco, CA, 1999.
- [14] Foster I., "The Globus Toolkit for Grid Computing," in *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001, pp. 2.
- [15] Foster I., Kesselman C., and Tuecke S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *The International Journal of High Performance Computing Applications*, Volume 15, Issue 3, June 2001, pp. 200-222.
- [16] Foster I., Kesselman C., Tsudik G., and Tuecke S., "A Security Architecture for Computational Grids," in *Proceedings of the 5th ACM Conference on Computer and Communication Security*, San Francisco, CA, November 1998, pp. 83-91.
- [17] Foster I., "What is the Grid? A three-point checklist," Argonne National Laboratory & University of Chicago, <http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf>, last accessed on October 22, 2005.
- [18] Frey J., Tannenbaum T., Livny M., Foster I., and Tuecke S., "Condor-G: A Computation Management Agent for Multi-institutional Grids," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, Aug. 2001, pp. 55-63.
- [19] Fujimoto N., Kenichi H., "A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks," in *Proceedings of the Symposium on Applications*

and the Internet-Workshops (SAINT 2004 Workshops), Tokyo, Japan, January 2004, pp. 674-680.

[20] He X., Sun X.H., and Laszewski G., "A QoS Guided Scheduling Algorithm for the Computational Grid," *Journal of Computer Science and Technology*, Volume 18, Issue 4, 2003, pp. 442-451.

[21] Howell F., McNab R., "SimJava: A Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling," in *Proceedings of the 1st International Conference on Web-based Modelling and Simulation*, San Diego, CA, Society for Computer Simulation, 1998.

[22] Howes T.A. and Smith M.C., "A scalable, deployable directory service framework for the internet," *Technical Report CITI TR-95-7*, Center for Information Technology Integration, University of Michigan, Ann Arbor, MI, July 1995.

[23] Howes T.A., "The Lightweight Directory Access Protocol: X.500 Lite," *Technical Report CITI TR-95-8*, Center for Information Technology Integration, University of Michigan, Ann Arbor, MI, July 1995.

[24] Iamnitchi A., Foster I., and Nurmi D.C., "A Peer-to-Peer Approach to Resource Location in Grid Environments," in *Proceedings of the 11th Symposium on High Performance Distributed Computing*, Edinburgh, UK, Aug. 2002, pp 419.

[25] Iyengar V., Sameer T., Lewis M.J., and Abu-Ghazaleh N.B., "Non-Uniform Information Dissemination for Dynamic Grid Resource Discovery," in *Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications (NCA 2004)*, Cambridge, MA, August-September 2004, pp. 97-106.

[26] Jacob B., "Grid Computing: What are the key components," IBM Inc., <http://www-128.ibm.com/developerworks/grid/library/gr-overview/>, last accessed on October 22, 2005.

[27] Maheswaran M. and Krauter K., "A Parameter-Based Approach to Resource Discovery in Grid Computing System," in *Proceedings of the 1st IEEE/ACM international Workshop on Grid Computing*, Bangalore, India, December 2000, pp.181-190.

[28] Maheswaran M., Ali S., Siegel H., Hensgen D., and Freund R.F., "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, Volume 59, Issue 2, November 1999, pp. 107-131.

[29] Maheswaran M., Ali S., Siegel H.J., Hensgen D., and Freund R.F., "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous

Computing Systems,” in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*, Washington, DC, April 1999, pp. 30-44.

[30] McLaughlin Michael P., “Compendium of Common Probability Distributions,” http://www.causascientia.org/math_stat/Dists/Compendium.pdf, last accessed on October 22, 2005.

[31] Moore J., “Grid Computing,” FCW, <http://www.fcw.com/article89311-06-20-05-Print>, last accessed on August 22, 2005.

[32] Paranhos D., Cirne W., and. Brasileiro F., “Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids,” in *Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003)*, Klagenfurt, Austria, August 2003, pp.169-180.

[33] Raman R., Livny M., and Solomon M., “Matchmaking: Distributed Resource Management for High Throughput Computing,” in *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing*, Chicago, IL, July 1998, pp.140-146.

[34] Schopf J.M., “A General Architecture for Scheduling on the Grid,” *Technical Report ANL/MCS-P1000-10002*, Argonne National Laboratory, Argonne, IL, April 2002.

[35] Smith D., “A Whirlwind Introduction to the Internet,” Department of Computer Science, University of North Carolina, <http://www.cs.unc.edu/Courses/comp123-s05/>, last accessed on November 7, 2005.

[36] Thompson R. and Clavenna S., “Grid Networking,” Byte and Switch Storage Networking and Beyond, <http://www.byteandswitch.com/>, last accessed on October 22, 2005