

# **Data Intensive and Network Aware (DIANA) Grid Scheduling**

**Ashiq Anjum**

A dissertation submitted in partial fulfilment of the requirements of  
The University of the West of England, Bristol for the degree of  
Doctor of Philosophy

This research programme was carried out  
in collaboration with CERN, Geneva

**The Faculty of Computing, Engineering and Mathematical Sciences,  
University of the West of England, Bristol 2006**

July 2007

## Abstract

*In scientific environments such as High Energy Physics (HEP), hundreds of end-users may individually or collectively submit thousands of jobs that access subsets of the petabytes of HEP data distributed over the world. Given the large number of jobs that can result from the splitting process and the amount of data being used by these jobs, it is possible to submit the job clusters (batch of similar jobs) to some scheduler as a unique entity, with subsequent optimization in the handling of the input datasets. In this process, known as bulk scheduling, jobs compete for scarce compute and storage resources and this can distribute the load disproportionately among available Grid nodes. Moreover, the Grid scheduling decisions are often made on the basis of jobs being either data or computation intensive: in data intensive situations jobs may be pushed to the data and in computation intensive situations data may be pulled to the jobs. This kind of scheduling, in which there is no consideration of network characteristics, can lead to performance degradation in a Grid environment and may result in large processing queues and job execution delays due to site overloads. Furthermore, previous approaches have been based on so-called greedy algorithms where a job is submitted to a 'best' resource without assessing the global cost of this action. However, this can lead to a skewing (unproportional distribution) in the distribution of resources and can result in large queues, reduced performance and throughput degradation for the remainder of the jobs. In this thesis we investigate a Data Intensive and Network Aware (DIANA) meta-scheduling approach which takes into account a cost based mechanism to map jobs against the resources when making scheduling decisions across multiple sites. We also present an extended version of the DIANA scheduling system which not only allocates best available resources to a job but also checks the global state of jobs and resources so that the output of the whole Grid is maximized and no single user or job can undergo starvation. DIANA is a performance-aware and an economy-guided Meta Scheduler. The DIANA meta-schedulers create a peer-to-peer hierarchy of schedulers to accomplish resource management, since existing scheduling hierarchies are not sufficient for Grid systems due to their inability to change with evolving loads and due to the dynamic and volatile nature of the resources. We detail the DIANA scheduling algorithm and its queue management system for coping with the load distribution and supporting bulk job scheduling. Results indicate that considerable performance improvements can be gained by adopting the DIANA scheduling approach.*

## Acknowledgements

I finish my thesis with small anecdotes in my acknowledgement section. Many friends and colleagues have contributed to my success, and I want to thank them all here.

First, I would like to thank my director of studies Prof. Richard McClatchey for his full support, expert guidance, wise counsel and understanding throughout my study, and for the generous funding which enabled me to travel, discuss and share my work with experts all around the world. I am fortunate enough to work with such a wonderful supervisor who is not only an excellent academic mentor but he developed my personality in a number of ways through his utmost patience, healthy criticism, vivid encouragement and occasionally refreshments.

Second, I am deeply indebted to my supervisor at CERN Prof. Ian Willers for his time, support and professional assistance in the development and completion of this research, and the European Centre for Nuclear Research for giving me an opportunity to be part of an exceptional research environment.

Third, I highly acknowledge the marvelous support, encouragement and counseling from Prof. Arshad Ali who is also my third supervisor. He always supported me financially, morally, technically and even psychologically when I used to be in trouble, always helped like an elder brother and father. He introduced me to the deep sea of research and I can still remember the mid-night video conferences with Caltech to sort out the research related issues in the early days of my research in 2002.

I should thank Heinz Stockinger from the Swiss Institute of Bioinformatics for his very apt technical assistance, friendly guidance and useful discussions throughout the course of this research. He is a wonderful friend who helped me a number of times, in a number of ways and facilitated my research in the best possible way. His valuable technical input and review on papers always provided very helpful insight.

There are many other important contributions from a number of important people which need due acclamation. Dr. Julian Bunn from CACR and Prof. Harvey Newman from Caltech deserve my special thanks for their technical input, supervision, guidance and support. I am very thankful to Michael Thomas, Conrad Steernberg and Frank Van Lingen from the Caltech team for their technical discussion on many issues and support during the work on the Clarens framework. Michael helped a lot to make JClarens a reality and extended his support during the work on the Data Location Service.

I should also thank to Iosif Legrand from Caltech for his assistance on MonaLISA and MONARC related issues. At the same time I should be grateful to Les Cottrell from SLAC for providing valuable insight in the use of the PingER performance monitoring tool for Grid Scheduling. I should be thankful to Lucas Taylor, Laasi Turra and Shahzad Muzaffar from North Eastern American University for the review of my proposals, reports and papers and then providing their valuable comments to use them in the CERN environment and Tony Johnson from SLAC for helping to port the JAS software. I should also thank to Erwin Laure and Florida Estrella from EGEE, Carl Kassleman, Steve Tucke and Charles Bacon from Globus, Miron Livny and Alain Roy from Condor, Richard Cavanaugh and Jang In from Sphinx and UFL, Geoffrey Fox for discussions on NARADA brokering, Lucia Silvertris and Vipin Bhatnagar from CERN, Akram Khan from SLAC, Roberto Barbera from INFN, David Wallom from Oxford, Miguel Marquina and Francois Fluckiger from CERN, Asif Jan from EPFL, Rene Brun and Peter Elmer from the Root team for their assistance on number of technical issues. UWE, CERN, NIIT, Caltech and other team members also deserve my heartiest thanks and space does not permit me to acknowledge their valuable services.

I will not be doing justice if I cannot acknowledge the tireless services, care and support of my wife Ammara throughout my research work. Her patience, countless sacrifices and understanding remained a prime catalyst for the timely completion of this work. I should also be very grateful to my parents who always remained curious and supportive for my research work and above all I cannot forget their countless prayers for my success.

Ashiq Anjum  
July 2006 CERN Geneva Switzerland.

## Table of Contents

<b>Abstract .....</b>	<b>2</b>
<b>Acknowledgements .....</b>	<b>3</b>
<b>Table of Contents .....</b>	<b>4</b>
<b>Chapter 1 Introduction .....</b>	<b>8</b>
1.1 Introduction .....	8
1.1.1 Grid Systems .....	9
1.1.2 Meta-Scheduler .....	10
1.1.3 Bulk Scheduling .....	11
1.2 Problem Description .....	12
1.3 Research Hypothesis and Research Questions .....	15
1.4 Work Performed in this Programme of Research .....	16
1.5 Thesis Organization .....	18
<b>Chapter 2 Current Research in Grid Scheduling and Resource Management .....</b>	<b>20</b>
2.1 Grid Resource Management Systems (GRMS) .....	20
2.1.1 The Grid Scheduling Process .....	21
2.1.2 Scheduling Steps .....	22
2.1.3 Meta versus Local Scheduling .....	23
2.2 Data-Intensive Resource Management .....	24
2.2.1 Data-Aware Scheduling .....	24
2.2.2 Replica Location and Scheduling Optimization .....	25
2.2.3 Data Schedulers and Meta Scheduling .....	25
2.2.4 Impact of Data Location on Scheduling .....	26
2.3 Network Aware Scheduling .....	27
2.3.1 Bandwidth Factor .....	28
2.3.2 Monitoring and Migration .....	29
2.4 Scheduling Optimization .....	30
2.4.1 Scheduling Optimization across Heterogeneous Systems .....	31
2.4.2 Application Level Scheduling .....	32
2.4.3 Economy Aware Scheduling .....	32
2.5 Batch Scheduling and Execution Systems .....	33
2.6 Meta Scheduling Algorithms .....	34
2.6.1 Traditional Scheduling Algorithms .....	34
2.6.2 Scheduling Optimization Algorithms .....	35
2.6.3 Global Cooperative Scheduling Algorithms .....	36
2.6.4 Priority Based Scheduling Algorithms .....	36
2.7 Meta Scheduling and Grid Standards .....	37
2.7.1 Scheduling Architecture .....	38

2.7.2	Scheduling and Resource Management .....	38
2.7.3	Scheduling and Monitoring .....	39
2.8	Conclusions.....	40
<b>Chapter 3</b>	<b>DIANA Grid Scheduling Requirements.....</b>	<b>41</b>
3.1	Introduction .....	41
3.2	Requirements Engineering Process for DIANA Grid Scheduling.....	43
3.3	Requirements Extraction from the Use Case Model.....	44
3.4	Analysis of the Requirements .....	52
3.4.1	Compute-Related Requirements .....	52
3.4.2	Data-Related Requirements .....	53
3.4.3	Network-Related Requirements.....	55
3.5	The Impact of the Use-Case Approach and Requirement Recommendations .....	56
3.6	Conclusions.....	58
<b>Chapter 4</b>	<b>Scheduling Optimization Approaches .....</b>	<b>60</b>
4.1	Introduction .....	60
4.2	Input Parameters and Objectives.....	61
4.3	Cost Estimators.....	62
4.3.1	Network Cost .....	63
4.3.2	Computation Cost.....	66
4.3.3	Data Transfer Cost .....	67
4.3.4	Total Cost .....	69
4.3.5	Allocation of Weights .....	69
4.4	Scheduling Algorithm .....	72
4.4.1	Pseudo Code of the Algorithm.....	72
4.4.2	Scheduling Matrix .....	75
4.4.3	An Example Scheduling Matrix.....	78
4.5	Queue Management in DIANA.....	80
4.5.1	Multilevel Queue Scheduling .....	80
4.5.2	Queue Management Algorithm.....	82
4.6	Bulk Scheduling .....	87
4.6.1	Bulk Scheduling with DIANA.....	88
4.6.2	Priority and Bulk Scheduling.....	88
4.6.3	Bulk Scheduling Algorithm Characteristics .....	89
4.6.4	Bulk Scheduling Algorithm.....	94
4.7	Job Migration Algorithm .....	97
4.8	Conclusion .....	99
<b>Chapter 5</b>	<b>Scheduling Hierarchies and DIANA Architecture.....</b>	<b>100</b>
5.1	Introduction .....	100
5.2	Considerations for Scheduling Hierarchies .....	103

5.2.1	Performance .....	103
5.2.2	Reliability .....	103
5.2.3	Network Scalability .....	104
5.2.4	Scheduling Policies .....	104
5.3	Scheduling Architectures .....	105
5.3.1	Master/Agent Architecture .....	105
5.3.2	Push and Pull Model.....	106
5.3.3	A Peer-to-Peer (P2P)-based Cooperative Scheduling Architecture .....	107
5.4	Hierarchies of Schedulers .....	107
5.4.1	Meta-Scheduling with DIANA .....	109
5.4.2	Queue Management.....	112
5.5	DIANA Architecture .....	113
5.5.1	General Architecture.....	114
5.5.2	DIANA's Scheduler Interface.....	117
5.6	Conclusions.....	120
<b>Chapter 6 Network Aware Meta-Scheduling .....</b>		<b>122</b>
6.1	Introduction .....	122
6.2	Network Measurements and Meta-Scheduling .....	124
6.3	Monitoring Infrastructure for the DIANA Scheduling .....	126
6.4	Network and Information Discovery .....	128
6.4.1	Discovery Service and Scheduling .....	130
6.4.2	Architectural and Design Characteristics.....	130
6.5	Peer-to-Peer (P2P) Topology-Aware Discovery .....	133
6.6	Implementation and Experimentation Details.....	137
6.7	Conclusion .....	140
<b>Chapter 7 Data Management and Scheduling Optimization .....</b>		<b>141</b>
7.1	Introduction .....	141
7.2	Data Management and Scheduling.....	142
7.2.1	Virtualization and Scheduling .....	143
7.2.2	Co-Scheduling and the Data Scheduler .....	145
7.2.3	The Catalogs and the Scheduling Process .....	146
7.3	Data Location Service .....	147
7.3.1	Architectural and Implementation Details .....	150
7.3.2	Data Location Interface (DLI) .....	152
7.4	Conclusions.....	153
<b>Chapter 8 Results and Discussion.....</b>		<b>155</b>
8.1	Introduction .....	155
8.2	Experimental Results through the GILDA Testbed.....	157

8.3	DIANA P2P Scheduling Results.....	166
8.3.1	Execution and Queue times with DIANA on the Custom Testbed.....	167
8.3.2	Discovery and propagation performance in a Peer to Peer Network .....	170
8.3.3	Job Migration and Monitoring .....	173
8.4	Results for Bulk Scheduling .....	177
8.4.1	MONARC Simulations for Bulk Scheduling .....	178
8.4.2	Scalability and Consistency Tests .....	186
8.5	Conclusions.....	194
<b>Chapter 9 Summary and Future Directions .....</b>		<b>196</b>
9.1	Summary.....	196
9.2	Critical Analysis and Conclusions .....	198
9.3	Future Direction.....	204
9.3.1	Potential Limitations of the DIANA Algorithm .....	204
9.3.2	The Comparison of Bulk Scheduling Algorithms .....	205
9.3.3	Network Managed Scheduling.....	205
9.3.4	From Grid Middleware to a Grid Operating System .....	206
9.3.5	Pre-emptive Scheduling of the Bulk Jobs.....	207
9.3.6	Meta-Scheduling Peers Grouping .....	207
9.3.7	Workflow Optimization through DIANA .....	208
9.4	Closing Statement.....	208
<b>Bibliography.....</b>		<b>209</b>
<b>Appendix-A</b>	<b>Bulk Scheduling Algorithm .....</b>	<b>222</b>
<b>Appendix-II</b>	<b>List of the Publications .....</b>	<b>224</b>

## Chapter 1

### Introduction

#### 1.1 Introduction

In this thesis I report on Data Intensive and Network Aware (DIANA) Grid scheduling. The goal of this thesis is to explore the combined impact of network characteristics, computing resources and data intensive jobs on the scheduling decisions when the resources are distributed over the global Grid. In Grids, for example LCG computing Grid [174] and Open Science Grid [173], scheduling decisions are often made on the basis of jobs being either data or computation intensive: in data intensive situations jobs may be pushed to the data and in computation intensive situations data may be pulled to the jobs. This scheduling approach can lead to performance degradation in a Grid environment and may result in large processing queues and job execution delays due to site overloads.

In the scientific analysis environment, a large number of tasks need to access Grid resources and users can submit a number of jobs which consume and produce a great deal of data that is potentially distributed worldwide. All these jobs will compete for scarce resources and this is likely to distribute the load disproportionately among the Grid nodes. Moreover, current scheduling systems [5] [6] [8] are inflexible to changing schedules and behave like static time-dependent Grid systems. These schedulers cannot meet the input constraints such as network characteristics and data location at runtime. The job scheduler should take into consideration input parameters such as data location, data size, site availability, network characteristics, computation power and different optimization parameters in making scheduling decisions.

In this thesis we describe a Data Intensive and Network Aware (DIANA) meta-scheduling approach which takes into account data, processing power and network characteristics when making scheduling decisions across multiple sites. The DIANA approach considers the Grid as a combination of active network elements [171] (to include the network as an integral Grid component that offers reliable, and if possible guaranteed, levels of service) and takes network characteristics as a first class criterion in the scheduling decision matrix, along with computations and data. The scheduler can make “intelligent” decisions by taking into account

the changing state of the network, the locality and the size of the data and the pool of processing cycles.

### 1.1.1 Grid Systems

Grid systems [1] [2] are interconnected collections of heterogeneous and geographically distributed resources harnessed together to satisfy the various needs of users. Grids can be classified into Computational Grids and Data Grids:

- Computational Grids, for example TeraGrid [172], are used for compute intensive operations and apply the resources of many computers in a network to a single problem that requires a large number of computer processing cycles for massive throughput and enhanced computing capability.
- Data Grids [173][174], on the other hand, support data intensive operations and promise to build the next generation computing infrastructures by providing intensive computation and analysis of shared large-scale datasets, from hundreds of Terabytes to Petabytes, across widely distributed scientific communities. This process involves a secure, reliable data transport protocol and a replica management system which replicates data to geographically distributed storage repositories to achieve high-performance data access.

Resource management is a central task in any Grid system. The Grid's basic responsibility is to accept requests from users, match these requests to available resources for which the user has access and then schedule execution using the matched resources. Resources include "traditional" resources like compute cycles, network bandwidth and storage systems. Typical resource management systems are Globus GRAM [5], WMS [6] from EDG, SGE [7], Condor [8] and the EuroGrid-Unicore [9] resource broker. Effective resource management and scheduling is a challenging issue, and data location and networks in addition to the computing power are critical factors in making scheduling decisions. The quality and consistency of networks are among the most important factors in this whole scheduling paradigm since the Grid can be subject to failure if networks do not perform well. Moreover, a site that has the targeted data may not be the best place for computation even if it has sufficient available computing power since these processors might be required to wait for a long time to fetch remote data from the storage media. Similarly, a site with the required data

may not be a good place to perform the computation if it does not have sufficient available computational resources. All these parameters need to be considered while making efficient scheduling decisions. Grid applications are becoming more and more network dependent with more demanding requirements in areas such as data access or interactivity, both with user tasks and other tasks. The specific kind of tasks that request computations are usually referred to as “jobs” and are dependent on storage, network capacity and computation. A job is an application or task performed on High Performance Computing resources. A Job may be composed of steps/sections as individual schedulable entities. When a job is submitted to a Grid scheduling system, the scheduling system has the responsibility to select a suitable resource and then manage the job execution. The system will then pass the job to an appropriate computing resource (often referred to as a Computing Element (CE) [10]) for execution, taking into account the requirements and the preferences expressed in the job description. The determination of which resources should be used is the outcome of a matchmaking process between submission requests and the available resources. The availability of resources for a particular task depends not only on the state of the resources, but also on the utilization policies that have been put in place by the resource administrators and/or the administrator of the Virtual Organization (VO) to which the user belongs.

### **1.1.2 Meta-Scheduler**

A meta-scheduler, as shown in Figure 1.1, facilitates the requesting of resources from more than one site for single or bulk jobs (see section 1.1.3) and performs load balancing of workloads across multiple sites. The fundamental difference between a meta-scheduler and local schedulers is that a meta-scheduler does not own the resources and has no autonomy in its decisions. Each site also has its own local scheduler to determine how its job queue is processed. The meta-scheduler maintains a global queue in which jobs waiting to be allocated to sites are queued for a certain time. A meta-scheduling system works on the basis that the “new task” which needs to be executed has to make itself known to a so-called “matchmaker”. This matchmaker acts as a gateway to the Grid. It selects resources from a global directory (e.g. an information service) and then allocates the job to one of the available Grid sites. Typically, this job allocation is done in two stages. Firstly a job is allocated to a particular site on the Grid by a meta-scheduler and then within that site, the job will be scheduled onto the individual processors by a Local Resource Management System (LRMS).

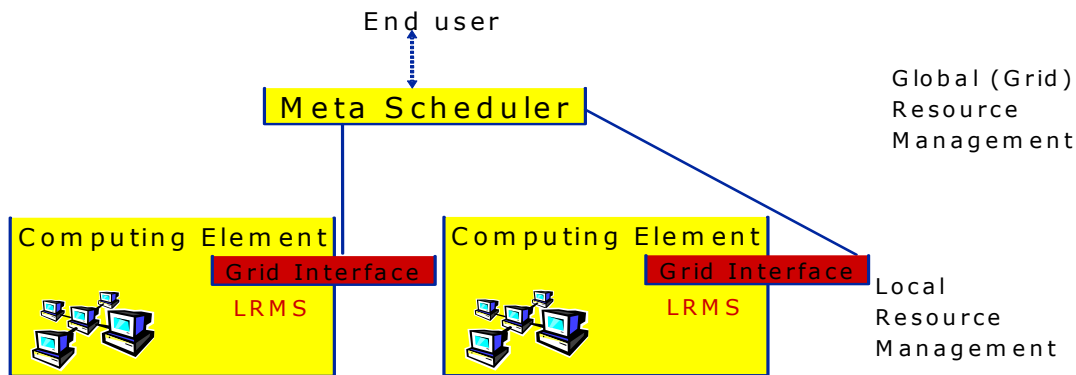


Figure 1.1: A Grid Meta-Scheduler interacting with Local Resource Manager Systems

### 1.1.3 Bulk Scheduling

In Scientific Analysis Environments such as the High Energy Physics (HEP), hundreds of end users submit (individually or collectively) thousands of jobs that access some subset of data distributed all over the world, this process being known as the bulk submission. The similar jobs are composed in a single bundle and are called the bulk jobs. To improve the scheduling and execution process, large jobs are divided into smaller jobs and the resulting sub-jobs can be submitted on one or more sites. Given the possibly large number of jobs resulting from the job-splitting procedure, it should be possible to submit the bulk jobs, in the form of job clusters, to the scheduler as a unique entity, with optimization in the handling of the input and output data. For example, the CMS physicist normally runs the complete analysis in parallel by submitting hundreds or thousands of jobs accessing different data files. A job generally consists of many subjobs [111] and some large jobs might even contain tens of thousands of subjobs which can start and run in parallel. Subjobs do not communicate with each other directly using any inter-process communication layer (such as MPI). Instead all data is passed, asynchronously, via datasets. Consequently if the data is concentrated on a single service, then this places a large burden on that service and the network to that service and this necessitates a special scheduling mechanism. A subjob generally has one or more datasets as its input, and will generally create or update at least one dataset to store its output.

Presented below are the estimates [128] for the typical number of jobs from users and their computation and data related requirements which should be supported by the CMS Grid.

- *Number of simultaneously active users: 100 (1000)*
- *Number of jobs submitted per day: 250 (10,000)*
- *Number of jobs being processed in parallel: 50 (1000)*
- *Job turnaround time: 30 seconds (for tiny jobs) – 1 month (for huge jobs) (0.2 seconds - 5 months)*
- *Number of datasets that serve as input to a subjob: 0-10 (0-50)*
- *Average number of datasets accessed by a job: 107 (250,000)*
- *Average size of the dataset accessed by a job: 30GB (1-3 TB)*

Note that the above parameters have a wide range of values, so that simple averages are not very meaningful in the absence of variances. For each parameter, the first value given is the expected value that needs to be supported as a minimum by the Grid system to be useful to CMS. The second value, in parentheses, is the expected value that is needed to support maximum levels of usage by individual physicists. Given these statistics about workloads, it is clearly challenging to intelligently schedule jobs and to improve resource usage over the Grid. All jobs will compete for the scarce resources and this is likely to distribute the load disproportionately among the Grid nodes. A scheduling system is required which not only allocates the best available resources to a job but also checks the global state of all the jobs and resources so that the strategic output of the whole Grid is maximized and no single job or user can consume the whole resources. This has led us to consider a bulk scheduling approach for tackling such distributed analysis scenarios.

## **1.2 Problem Description**

Traditionally, in the case of data intensive jobs, scheduling decisions have been made by moving the executables to the data and in the case of computation intensive jobs by moving data to the executables. Data intensive applications, through millions of jobs submitted individually or in bulk, analyze large amounts of data which are replicated to geographically distributed sites. If data are not replicated to the site where the jobs are supposed to be executed, the data needs to be fetched from remote sites. This data transfer from other sites will definitely degrade the overall performance of the job execution. If a computing job runs at a remote site, the produced output data need to be transferred to the user so he can analyze the result locally. In order to support this type of phenomenon, the Grid has a high number of

Computing Services (CE) and Storage Services (SE) (see figure 1.2) each of them identified by a unique ID through an information service.

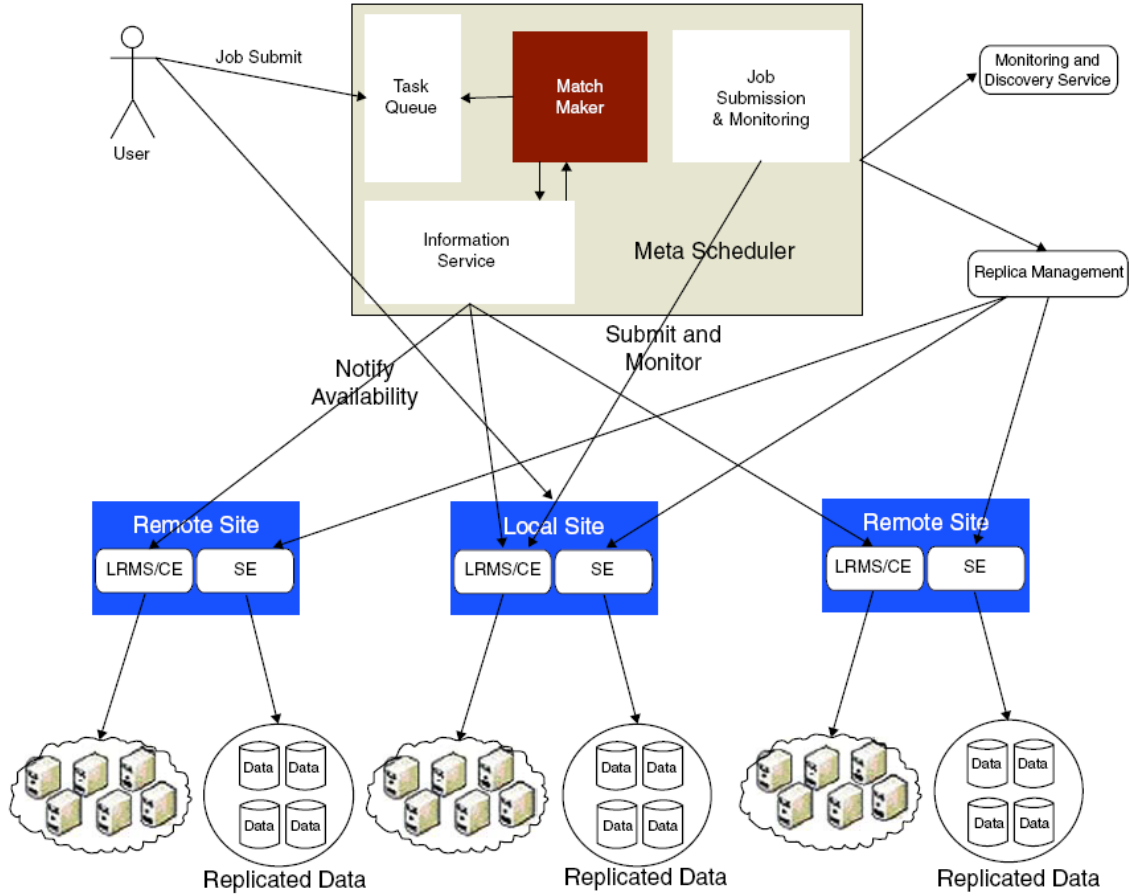


Figure 1.2: Multiple sites Grid Meta-scheduling

Each data intensive job produces and consumes different amounts of data. For performance gains in the overall job execution time and to maximize the Grid throughput, we need to align and co-schedule the computation and the data (input as well as output) in such a way that we can reduce the overall computation and data transfer cost. We may even decide to send both the data and the executables to a third location depending on the capabilities and characteristics of the computing, network and storage resources. This is a simple problem if taken from a local resource management point of view but can be challenging from a meta-scheduling context where it is intended to optimize the overall Grid throughput. We consider only meta-scheduling, a process which allows a user to schedule a job across multiple sites. As we add more complexity to the Grid, particularly with geographically dispersed sites or nodes and with increasing number of jobs, it becomes common for the meta-scheduler and

data mover (or data transfer service) to make decisions which may be in conflict with decisions that would be arrived at from a local scheduling point of view. This is not surprising since, originally, many of the scheduling technologies were developed under the assumption that all the collaborating systems were on the same local area network (LAN). If we extend these scheduling or local resource management systems over longer distances, we can begin to see the limitations of the global decision-making systems.

The Grid will have a number of computational sites which are connected by a number of different WAN links. These links will normally have different bandwidths and latencies. We require the Grid to support a number of varied workloads especially for the bulk scheduling process, and this, in combination with the network requirements, leads to the likelihood that some executables and data items will be large compared to the available network bandwidths and latencies. Given these constraints on the overall system, we need to find a way to distribute workloads across all the available systems to maximize the utilization of the available systems. The costs of performing computation will vary according to the types of machines used, the network bandwidth consumed, the number of jobs and their frequency, the state and reliability of the network system, the losses and routing issues in the networks, and will also depend on other factors, such as the amount and the location of the data required. These parameters need to be taken into account when work is scheduled to ensure the effective use of resources to minimize cost and yet maximize workload throughput.

Hence, the centralized scheduling algorithms that focus only on maximizing processor utilization by mapping jobs to idle processors, and/or on disregarding network costs, queue times, job priorities and costs associated with accessing remote data are unlikely to be efficient [17]. Similarly, the scheduling decisions which always force the job movement towards the data without taking the Grid “weather”, the network load and the data size and location into consideration can lead to significant inefficiencies in performance and can be responsible for large job queues and processing delays. Considering the effects of and compensating for network characteristics can avoid making these less-than-ideal scheduling decisions. We not only need to use the network and other characteristics while aligning data and computations but we also need to optimize the task queues of the meta-scheduler on the basis of this correlation so that these characteristics can play an important role in the matchmaking process and on Grid scheduling optimization. Thus, a decentralized meta

scheduling mechanism is required that should consider the job execution, queue statistics, priorities and data transfer and their relation with various network parameters collected through different monitoring tools [11] [12] on multiple sites, so that we can ascertain the performance of the jobs on the basis of these characteristics and hence can optimize the meta-scheduling [13] [14]. Our big challenge then becomes finding a means to express these requirements in a format that the meta-scheduler engine can understand. This engine should use mathematical techniques to make decisions and to generate the overall behaviour of the system based on the global network characteristics in local environments.

The basic job scheduling algorithm at each site should be driven by a weighting value calculated for each potential target location which is a function of the available network characteristics, the processing cycles, job priorities, queue length and input and output “sandboxes” of data and the one having least cost should be given priority [175]. This should also consider the Grid as a combination of the active network elements and must take the network as a first class criterion in the scheduling decision matrix. Both the scheduling of resources and/or the moving of data from place to place as needed, as well as overseeing the task execution through to completion, need to be performed on the basis of a "network and strategic view" of the overall Grid system. This should ensure an optimized distribution of work and the Meta-Scheduler should then make intelligent decisions by taking into account the changing state of the Grid network, locality and the size of the data, number of jobs and the pool of processing cycles. The obvious impact from this approach should bring the network and Grid applications closer and should therefore help further to understand the needs of each other which evolved independently.

### **1.3 Research Hypothesis and Research Questions**

The research hypothesis investigated in this thesis asserts that:

Data intensive bulk scheduling can be significantly improved by taking into consideration a combination of network, data and compute costs, as well as by implementing effective queue management and priority control.

This thesis will therefore attempt to answers the following research questions:

- 1. What level of optimization is achieved when we co-allocate and co-schedule the compute intensive and data intensive jobs on the best ranked resources? What if data is moved towards jobs or both data and jobs are moved toward a third location?
- 2. How can network managed services and network characteristics optimize the data intensive Meta scheduling in the Grid?
- 3. What are the important bulk scheduling optimization strategies for the meta-scheduling process and how can such schedulers optimize the bulk scheduling? Can priority and queuing techniques play any role in the bulk scheduling optimization?
- 4. Can the inclusion of the network, data and compute costs in the scheduling decisions optimize the meta-scheduler queue and execution times?
- 5. Are centralized algorithms and environments less effective than their decentralized counterparts when scheduling data intensive bulk jobs? Can scheduling hierarchies have any impact on the scheduling optimization? Can the decentralized scheduling solutions scale at the global Grid level and are they also stable enough to accommodate large number of jobs and resources?

#### **1.4 Work Performed in this Programme of Research**

In this section, a list of the salient features of the research work done in this thesis is given. Although detailed description of the work will be presented in the following chapters, major tasks that have been performed and will be described in this thesis are outlined below.

- An adaptive scheduling algorithm “DIANA” has been created which takes the network, size and location of the data and compute information from various information resources and helps to select an optimum site for job execution. It also includes site loads and the site having the overall least cost is selected.
- A DIANA Scheduler has been created which implements the DIANA scheduling algorithm. This is a generic scheduler in which a wide variety of the algorithms can be implemented and performs matchmaking with the gLite resource broker. The DIANA Scheduler works with various network monitoring tools, information and directory services and replica catalogs to make optimal scheduling decisions.

- The issue of the bulk scheduling has been handled in the DIANA Scheduler. A comparative analysis of the simulation as well as experimental results is given. Moreover, the DIANA scheduling algorithm has been extended to accommodate an economy aware bulk scheduling for data intensive environments. Scalability tests have been conducted to test DIANA approach for bulk scheduling and the results are presented in chapter 8.
- Scheduling hierarchies were investigated and a peer-to-peer (P2P) scheduling architecture has been proposed for the bulk scheduling optimization. A detail analysis and architecture is discussed in chapter 5 and the related results are presented in chapter 8.
- A Discovery Service has been designed and discussed which enables the quick discovery of the resources and network information. This also facilitates the decentralized functioning of the DIANA meta-scheduler and chapter 6 discusses and covers the details of this Discovery Service. Results related to the Discovery Service are presented in the chapter 8.
- A Data Location Service has been created which selects the best replica of a data set from the available physical replicas. This decision is made on the basis of data location and network characteristics. This uses the Data Location Interface (DLI) which is created as a module of the EDG Work Load Management system [6]. The Data Location Service (DLS) is discussed in chapter 7.
- A Test setup was created on the EGEE GILDA [15] Testbed to test the DIANA Scheduler and its algorithms. The Data Location Service was also deployed on the GILDA Testbed.
- Results were taken from the test setup mentioned above and presented in journal and conference papers (As listed in Appendix-B). These results were based on the actual measurements taken from the experimental setup. These results depict the contribution of this research work.
- A simulation study of the points discussed above has been carried out. This provides an in-depth analysis of the results and helped to identify the issues which could not be highlighted due to limitation of the experimental setup. To establish a test setup and

then to make the tests on this large Grid testbed is lengthy and time consuming process. On the other hand, the simulations helped to find the extensive conclusions and provided a detailed image of the system and results.

## 1.5 Thesis Organization

In this thesis, we introduce a Data Intensive and Network Aware (DIANA) meta-scheduling approach which takes into account the network characteristics along with computation and data when scheduling single or bulk jobs. This section illustrates the thesis organization as shown in figure 1.3. In this Chapter 1, the nature of the research problem has been established and the issues involved in its study have been identified. In Chapter 2, the research background is illustrated and an overview of the literature survey is described. The projects which are active in this area of research as well as the current efforts of various standard bodies are also cited. A requirement analysis of the DIANA Meta Scheduling is provided in chapter 3 where a Use Case based methodology has been adopted to capture the requirements. Chapter 4 provides an overview and discusses the proposed scheduling optimization approaches and associated algorithms. A cost matrix, dependent on data, computation and network costs, has been created and approaches are given to populate and search the relevant costs, which then become part of the Meta scheduling process.

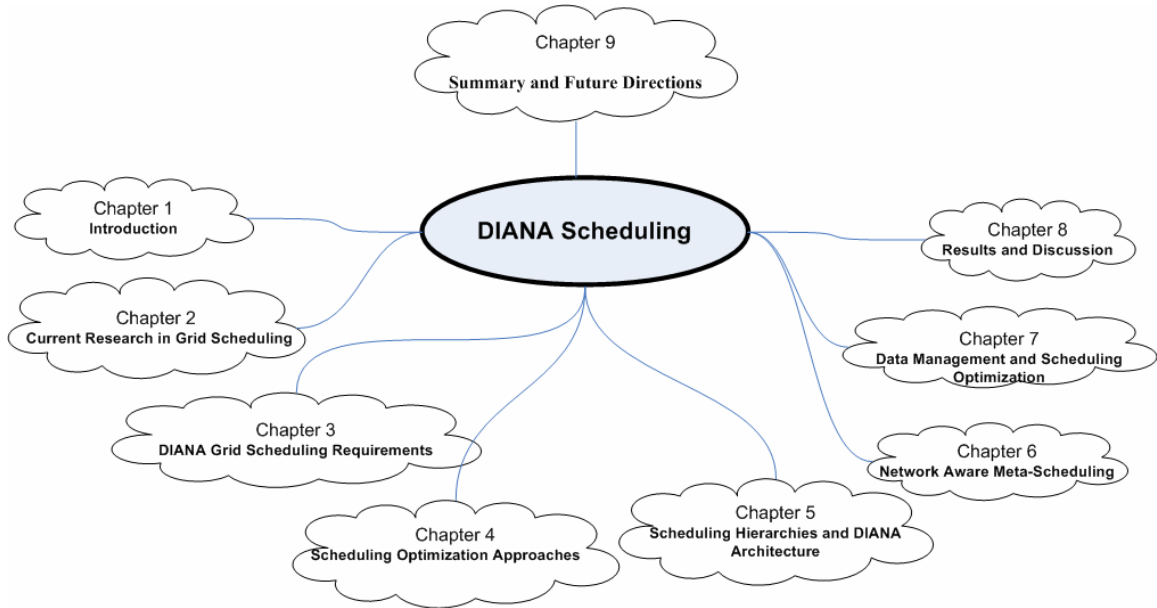


Figure 1.3: Thesis Organization

Chapter 5 describes the scheduling hierarchies for scheduling optimization and illustrates the architecture of the proposed system and highlights the components of this architecture. Chapter 6 describes the process to extract and analyze the network values and their impact on the scheduling decisions and explains the role of high performance networks and how network managed services influence data intensive scheduling decisions. It also discusses the discovery and information communication algorithm which facilitates the scheduler in the discovery and decentralized scheduling decisions. Chapter 7 describes the data related aspects of scheduling and illustrates the design and implementation of a Data Location Service (DLS). A detailed description of the results is given in Chapter 8. That chapter also describes the deployment of the system on a Grid testbed and gives details of the environment and issues involved with this experimental setup. Further simulation results of various approaches and algorithms are also presented in this chapter. Chapter 9 draws thesis conclusions and gives a critical and concise summary of the work that has been carried out, its application areas and its limitations and describes potential future extensions of the thesis work.

## Chapter 2

### Current Research in Grid Scheduling and Resource Management

The first chapter of this thesis introduced the research hypothesis that this thesis attempts to prove and the consequent research questions that it will attempt to answer. This chapter will survey the research undertaken in the thesis domain and will describe how it relates to the work of others. It will do this by identifying relevant background knowledge from previous research as well as describing alternative approaches to that undertaken in this thesis.

This chapter introduces the scheduling and resource management from the GGF [16] perspective and describes the components that comprise a Grid Resource Management System (GRMS). Since this study concentrates on data intensive and network aware Grid scheduling, this chapter gives an overview of research on data-aware scheduling and describes how network and bandwidth is a prime concern for data-intensive scheduling. A survey of the projects offering scheduling optimization in particular and meta-scheduling in general is also provided. This chapter identifies relevant research from other projects and activity in the field of standards.

#### 2.1 Grid Resource Management Systems (GRMS)

A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities as defined by Ian Foster et al. in their article [1]. Globus architects K. Czajkowski et al. state that resource management systems deal with managing a system and application tasks, involving security and fault tolerance along with scheduling [3]. R. Buyya from the university of Melbourne and others define Grid resource management as the process of identifying resource requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time, in order to run Grid applications as efficiently as possible [4]. Grid applications compete for resources that are very different in nature including processors, data, scientific instruments, networks, and other services. While Grids have become almost commonplace, the use of good Grid resource management tools is far from ubiquitous because of the many open issues that remain in the

field such as data intensive scheduling, cost based scheduling, network aware scheduling, the bulk scheduling, and decentralized and priority aware scheduling.

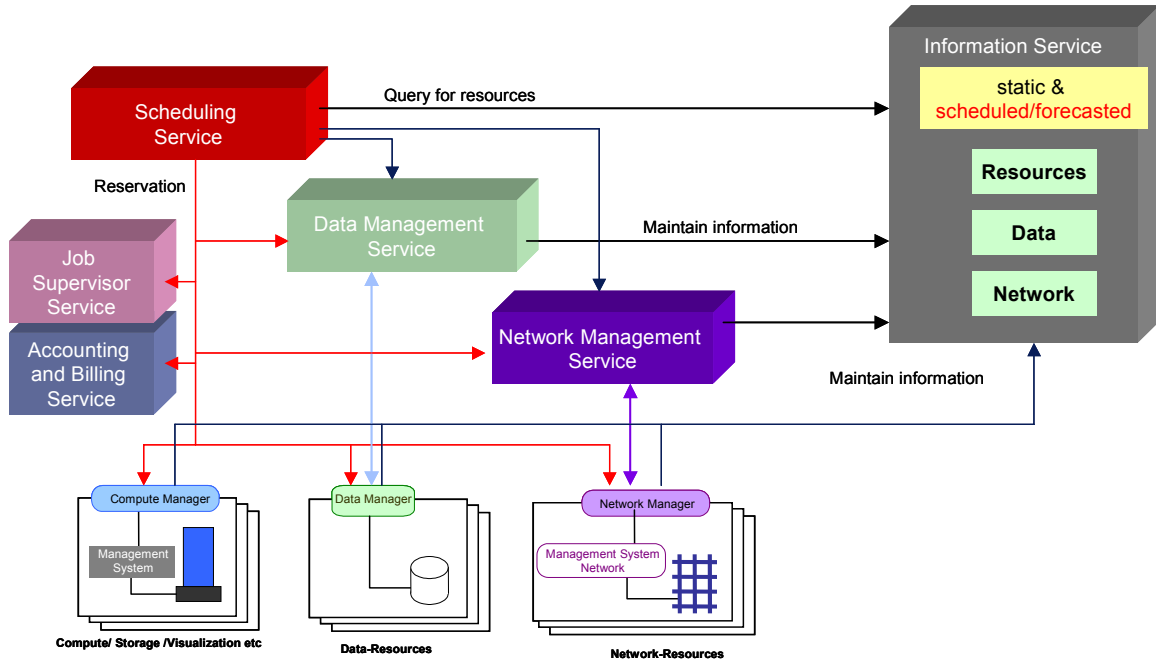


Figure 2.1: Services in the scheduling process (Adopted from the GGF scheduling area)

### 2.1.1 The Grid Scheduling Process

The Grid scheduler is a component (service) of the Grid Resource Management System and depends on the other services and components as shown in Figure-2.1. The Global Grid Forum (GGF) is a standard body for Grid related standards. The GGF scheduling working group has defined a standard for scheduling and a resource management architecture that supports cooperation between different scheduling instances for Grid resources [16]. The Grid Scheduler as shown in Figure-2.1 is one defined component. The Grid Scheduler discovers the resources available and chooses the best fit resources for the job. In order to filter unacceptable resources, the Grid Scheduler undertakes matchmaking between the resource specification in the job request and the resource owner's preference in the resource owner policy. The resource owner policy is delivered from the resource by the information service. Then the candidates selected from the matchmaking process enter the scheduling process and the final winners are picked out, based on the scheduling algorithm. The final process in Grid scheduling is the reservation of the resources which need to be scheduled in

future. Some of the components of the Grid scheduling service are discussed in the next section.

### **2.1.2 Scheduling Steps**

This section describes Grid scheduling steps as defined by the GGF scheduling working group and how the scheduling process works. There are three scheduling phases which are further divided into individual steps. Each step is performed by an autonomous component which coordinates and communicates with other components to perform the scheduling process. The first phase of the scheduling process deals with discovery of the distributed resources. To offer dynamic resources and to provide user convenience, a Grid scheduling service should be able to discover and select available resources from a Grid environment. This is managed by an Information service as shown in the Figure 2.1. The scheduling service is aided by this information service and this is normally accomplished in three steps: authorization filtering, job requirement knowledge and filtering to meet the minimal job requirements. The Grid environment is so dynamic and unpredictable that a Grid job should wait in the job queue until the scheduling process ends. A Queuing mechanism supports these basic scheduling capabilities. It allows administrators to customize existing policies and define new scheduling policies for their virtual organization. The jobs wait in a queue until a suitable resource is found on a particular site for their execution. Priorities and policies are used to manage the operation of the queue.

The second phase within the scheduling process is brokering and resource selection. Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This is generally done in two steps: information gathering which is the responsibility of the Information Service and decision making which is managed by the Scheduling Service. Scheduling is a process of matching a job to the appropriate resources. In this second phase, various scheduling algorithm could be applied. The Scheduling Service is further supported by Network Management and Data Management Services.

The third phase of meta-scheduling is the running of the job and this is performed by a job execution component. This involves a number of steps, few of which have been defined in a

uniform way between resources. Once resources have been chosen, the application must be submitted to the resources. This may be as easy as running a single command or as complicated as running a series of scripts, and may or may not include setup or staging. The Preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application. It may be the case that to make the best use of a given system, part or all of the resources will have to be reserved in advance by a Reservation component. Depending on the resource, this can be easy or difficult to achieve, but it may be done with mechanical means as opposed to human means, and the reservations may or may not expire with or without cost. The Monitoring of job status and resource status is performed once the job is being executed and this is managed by Job Supervisor Service.

Although resource monitoring is required to monitor and discover resources, it would be supported by an information service. Monitoring service facilitates the discovery of resources by identifying the available resources. Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing. Job Steering and control also lie under the functional boundaries of this step. The job execution is supported by a Compute Manager, Network Manager and a Storage Manager which are managed by a local scheduler which communicates with the global scheduling system.

### **2.1.3 Meta versus Local Scheduling**

Sebastian Ho and Eddy Caron [20][22] state that a meta-scheduler facilitates the requesting of resources from more than one machine for single or bulk jobs and performs load balancing of workloads across multiple sites. It coordinates communications between multiple heterogeneous schedulers that operate at the local or cluster level. In addition to providing a common entry point, a meta-scheduler also enables global access and coordination, whilst maintaining local control and ownership of resources. The fundamental difference between a Meta-scheduler and other common local schedulers is that a Meta-scheduler does not own the resources and has no autonomy in its decisions. Therefore it does not have total control over the resources. Furthermore, a Meta-scheduler does not have control over the entire set of jobs on the system, nor does it even necessarily know about them. So decisions about an entire set of jobs regarding some resource cannot be made by a Meta-Scheduler. This lack of

ownership and control are the sources of many of the problems to be solved in the Meta-scheduling domain.

## **2.2 Scheduling Aspects of Data-Intensive Resource Management**

This section gives an overview of the research progress in data management from the meta scheduling point of view. Data is an important aspect of the scheduling and the scheduler cannot take optimized decisions if data location, replica management and file transfer are not synchronized with the scheduling approach. According to Ranganathan and Foster et al. from Argonne National Labs [17], it is important for data-intensive jobs to take data location into account when determining job placement for these jobs. This increases the performance of the system and reduces the execution time of the jobs. Replication of data from primary repositories to other locations can also be an important optimization step, so as to reduce the frequency of remote data access. Since resources in the Grid are connected by wide area network links, bandwidth limitation is an issue that must be considered when running these applications on such environments. For these applications, the data reuse pattern can be exploited to achieve better performance. Here some important issues which discuss the data and scheduling relation are introduced.

### **2.2.1 Data-Aware Scheduling**

There are a number of methodological approaches that impact on scheduling performance. One such approach proposed by Cirne et al. from the MyGrid project is that of data-aware scheduling [18] [19], the process of scheduling tasks close to the data that they require. Baru et al. [21] prove that considerable performance improvements can be gained by taking into account the amount (and transfer rate) of data transfer that is required during a task execution. For example, this functionality is to be found in the JOSH (JOB Scheduling Hierarchy) [176] software, a new GT3-based hierarchical scheduler for the Sun Grid Engine. This system takes load and data proximity into account. Several computational Grid test beds are operational, but the challenges being faced by experiments at CERN have led to the concept of a petascale virtual-data Grid as described by Paul Avery et al. in their article [24]. "Petascale" emphasizes the massive CPU resources (Petaflops) and the enormous datasets (Petabytes) that must be harnessed, while "virtual" refers to data products that may not be physically stored but exist only as specifications as to how they may be derived from other

data. The central problem is the coordinated management of computation and data and not simply data movement.

### **2.2.2 Replica Location and Scheduling Optimization**

The Replica Location Service (RLS) provides a framework for tracking the physical locations of data that have been replicated. A number of research projects are under way which are tackling the Grid scheduling issues in general and optimization issues in particular. The Replica Optimization Service (ROS) as described by Bell et al. in [30] and by Cameron et al. [31] is the most related work that provides a functional system. The ROS selects a best location for replicas based on network and storage costs. The system was fully integrated with the European Data Grid's (EDG) workload management system and provided much of the functionality described here. However, the system used a "complicated" network monitoring infrastructure that was not further maintained within the EGEE/LCG projects and therefore the ROS was no longer deployed. Stockinger et al. [32] state that hierarchical storage systems are the main source of bottlenecks rather than network parameters. The GRESS project [33] is more like a framework into which various algorithms can be plugged to test their effectiveness. Moreover, Tan et al. [34] state performance results of various algorithms and give possible scenarios in which they can increase the scheduling efficiency. Nabrzyski et al. [35] outline an AI knowledge based Meta-Scheduler which performs a multi-criteria search technique while making scheduling decisions. Chervenak et al. [36] describe Gigggle, a framework to construct a decentralized, scalable and efficient replica location service. Ranganathan et al. [17] study the widely used dynamic replica strategies using simulation results. Their simulation results show that the job scheduling considering data locations shows promising performance. However none of the abovementioned approaches has considered the data intensive scheduling or treats the network as a resource in the scheduling decisions.

### **2.2.3 Data Schedulers and Meta Scheduling**

Data placement jobs should be treated differently from computational jobs, since they may have different semantics and different characteristics. For this purpose, a data scheduler for data placement activities in the Grid is required. The PhedEX project [37] at CMS [89] is a large-scale data staging, transfer and data scheduling environment which uses intelligent

agents to take individual data scheduling and transfer decisions and exploits the efficient bandwidth use while making data placement decisions. The Stork project [38] suggests data placement activities are equally important to that of computational jobs in the Grid so that data-intensive jobs are automatically queued, scheduled, monitored, managed, and even check-pointed as is done in the Condor project for computation jobs. Stork uses the capabilities of Condor for compute related jobs and suggests a way to associate the data placement activities with compute jobs while scheduling data-intensive jobs. Thain et al. [39] describe a system that binds jobs and data together by binding execution and storage sites into I/O communities. Basney et al. define an execution framework which provides an affinity between CPU and data resources in the Grid to run applications on the CPUs which have needed access to datasets. Although co-allocation is considered in their findings, network aware scheduling is not discussed at all. Moreover, none of the approaches considered bulk scheduling as a core scheduling problem and it is simply ignored in their discussion.

#### **2.2.4 Impact of Data Location on Scheduling**

Data location can be very important in making successful scheduling decisions. It may particularly be an issue in the case of data intensive environment and the complete schedule may fail if data location is not taken into account. Santos-Neto et al. state [41] that it is very important to take data transfer into account to achieve efficient scheduling of data-intensive applications on Grids since these are costly operations. Foster and Ranganathan suggest [17] that the traditional paradigm for scientific syntheses is to gather data at a single location and transform it into a common format prior to exploring it. They suggest that an effective optimization is to decouple data movement and computation so the data is staged to locations “near” (in terms of some access cost metric) to where it is required. Chameleon [42] implements a data Grid scheduler that takes into account both data location and processor cycles in its decision matrix but their algorithm is based on a ‘shortest response time first’ and instead in this thesis work we aim at a network-aware adaptive algorithm which takes dynamic decisions while scheduling data-intensive jobs. The greedy scheduling algorithms used by Chameleon have high resource cost and other shortcomings and this is one reason why we have used network-aware adaptive algorithms for our scheduling matrix.

### 2.3 Network Aware Scheduling

The network is the central point of any large distributed or Grid system and it is natural to include network characteristics in the scheduling systems. Network characteristics such as the bandwidth between the distributed sites, packet loss and round trip time (RTT) actually drive the modern day distributed systems and such characteristics can influence the scheduling efficiency by improving its schedule policy and selecting the reliable and better nodes for data transfer and job execution. L. Boloni and D. Marinesc describe Bond [51], a Java based object oriented middleware system for network computing. Bond has a two level scheduler based on a stock market or computational economy approach. It is mostly used for multimedia applications and has no real applications for data intensive problems. Chandra et al. from Carnegie Mellon University state that Darwin [52] is a customizable resource management system for creating value added network services; it also provides mechanisms for scheduling computation in non-network nodes. However it has not been designed for data intensive applications and its main use is in multimedia applications. The Globus project offers Grid information services via an LDAP-based network directory called MDS [53] which follows both push and pull protocols for resource dissemination. Although the Globus Meta Scheduler can be used for data intensive applications there is no mechanism in it by which it can measure and monitor the network in real time and can include it in the scheduling decisions.

Neary et al. from the University of California describe the Javelin [54] project. Javelin is a Java based project for internet-wide parallel computing and can be considered a computational Grid for high-throughput computing. Javelin provides solutions for the parallel applications and is ill-suited for network centric data intensive applications. The University of the Lancaster Distributed Multimedia Research Group (DMRG) has a Generic Object Platform Infrastructure (GOPI) project [55] based on CORBA with RM-ODP extension. It provides an extensible architecture for adaptive multimedia applications and forms a multimedia service Grid. The MOL [56] project at University of Paderborn is developing technologies that aim at utilizing multiple WAN-connected high performance systems as a computational resource for solving large-scale problems that are intractable on a single supercomputer. Its kernel offers basic generic infrastructure and core services for robust resource management that can be used to construct higher level services. The MSHN project

[57] at Purdue University is developing a resource management system for distributed heterogeneous environments. One unique aspect of MSHN is that it is targeted to support applications that can adapt to resource conditions in the Grid. Both MOL and MSHN are good for computation intensive applications but not suitable for data aware scheduling.

Netsolve [58] project of the Oak Ridge National Laboratory has produced a client-agent-server paradigm based network-enabled application server. Netsolve offers the ability to search for resources, to choose the best one available and to solve a problem with retry for fault-tolerance. Although it takes into account the network traffic in the scheduling matrix, it does not select the replicas on the basis of some optimal criteria and does not support data intensive scheduling. Kapadia and Figueiredo from Purdue University describe the work performed under PUNCH [59] project. PUNCH is a middleware Testbed that provides operating system services in a network-based computing environment. A network operating system (OS) layer provides distributed process management and data browsing services. It is more like a local scheduler, a browsing and data management system rather than a resource management system. This discussion indicates that although there are a number of scheduling solutions, none of them resolves the bulk scheduling problem in particular. Similarly, the network issues have not been adequately addressed in available scheduling products and tools.

Network aware scheduling techniques have been studied in practice by introducing QoS mechanisms that enable service providers to partition their services based on quality criteria such as priority, fairness, and economic gain [178]. But the problem under study is different to the multimedia applications. Although multimedia applications are dependant on the network characteristics to provide desired QOS, they are not as data intensive as is the case with Grid applications. Furthermore, in multimedia applications scheduling systems are not exposed to the same number of jobs as is the case with Grid applications.

### **2.3.1 Bandwidth Factor**

The key to deriving insight and knowledge is often the correlation of data from multiple sources as asserted by Ian Foster et al. [87] [96]. The traditional paradigm for such syntheses is to gather data at a single location and transform it into a common format prior to exploring it. However, the expense of this approach in terms of network resources has meant that most

data is never correlated or compared to other data as expressed by Beck and his colleagues from University of Tennessee [97]. One factor driving this new paradigm is the recent dramatic improvements in network performance. Few Internet networks move data at more than a megabit per second (Mbps) and would take weeks to move a terabyte. The term Bandwidth is used to describe network channel capacity, the rate at which bits may be transmitted through the system. Advances in networking technologies are ushering in a new era of bandwidth abundance based on terabytes per sec (Tbps) optical backbones that provide routine access to end-to-end paths of 10Gbps or more - an improvement of over four orders-of- magnitude. For example, in SC2005, a bandwidth record of 131 Gbps was achieved by Caltech and CERN teams [88]. This is critical for effective data integration and data intensive scheduling, since technologies are now beginning to allow distributed communities, or virtual organizations, to access and share data, networks, and other resources in a controlled manner. Therefore bandwidth is an important phenomenon and it can influence data intensive scheduling if it is included as a resource in scheduling decisions.

### **2.3.2 Monitoring and Migration**

A Grid is inherently a dynamic system where environmental conditions are subjected to unpredictable changes such as system or network failures, system performance degradation, the addition of new machines and variations in the cost of resources, etc. In such a context, job migration is the only efficient way to guarantee that the submitted jobs are completed and that user constraints are met. The major tasks involved in migration are job monitoring, re-scheduling, and checkpointing. Most of the systems dealing with job migration address the migration problem from the point of view of performance as described by Allen from Max-Planck-Institute et al. [27] and Huedo [28]. The main migration policies considered in these systems include, amongst others, performance slowdown, target system failure, job cancellation, detection of a better resource, etc. However, there are few systems that manage job migration under economic conditions like the one presented by Vadhiyar from the University of Tennessee et al. [29]. In this context, new job migration policies must be contemplated, such as the discovery of a new cheaper resource, or variations in the resource prices during the job execution. There are a broad variety of reasons that could lead resources to dynamically modify their prices, for example, prices can change according to demand. Prices can change according to the time or the day. For example, the use of a resource can be

cheaper during the night or during the weekend. GridSAM [177] is a job submission and monitoring web service for submitting and monitoring jobs managed by a variety of distributed resource managers. However GridSAM does not provide network aware scheduling and it depends on other scheduling tools to make a scheduling decision. It does not provide data intensive scheduling algorithms and there is no provision for the policy based scheduling.

## **2.4 Scheduling Optimization**

In this section descriptions of the projects that study the meta-scheduling frameworks and their association with underlying resource optimization issues are discussed. This section gives the current position and overview of the research in this area in terms of what has already been done or is currently being studied to optimize the scheduling process. Technically this is very rich area in research and a broad range of activities are being carried out in this domain. We are interested in those research projects which are dealing with data-intensive and network related issues of the scheduling and resource management and hence we list and give details of such selected projects and papers. One important stage of resource brokering which needs to be optimized is job scheduling, i.e., the mapping of pending jobs to specific physical resources, in an attempt to minimize some cost function specified by the user.

Rafael Moreno describes that schedulers can be classified into two major categories: performance-guided schedulers and economy-guided schedulers [95]. Most of the Grid systems in the literature fall in the first category, since they try to find a job-to-resource mapping that minimize the overall execution time (i.e. one which optimizes performance). One of the simplest performance-guided scheduling algorithms is the greedy or opportunistic approach, which iteratively allocates each job to the machine that is likely to produce the best performance, without considering the remainder of the pending jobs. This approach normally leads to sub-optimal solutions, since scheduling decisions are based only on local job information. Other more complex scheduling algorithms, which explore the solution space and try to overcome local optimal solutions, are based on genetic algorithms, simulated annealing [23], tabu search [24], branch and bound methods [25] or budget constraint (cost of resources) as in the Nimrod/G broker [26]. In the most general case, scheduling algorithms have to adapt to the different optimization criteria for each particular job [95].

Some of the most frequent optimization criteria that are important for a user are optimizing performance without regard to cost, optimizing cost without regard to performance, optimizing performance within a specific cost constraint, optimizing cost within a specific time constraint, optimizing performance within a specific cost and time constraint and optimizing cost within a specific cost and time constraint.

Although these algorithms do address a variety of scheduling issues they do not take care of data intensive scheduling based on network principles. Therefore we require adaptive algorithms which can dynamically adapt to the Grid network conditions and can optimize the scheduling process, an important consideration in the work presented in this thesis.

#### **2.4.1 Scheduling Optimization across Heterogeneous Systems**

A Meta Scheduler coordinates communications between multiple heterogeneous schedulers that operate at the local or cluster level. Grid level scheduling policies are managed by the Meta Scheduler whereas local control and ownership of resources comes under the jurisdiction of the local schedulers. The community scheduling project from Platform Computing [43] has created a meta-scheduler framework which provides a consistent interface for users into the Grid scheduling system. This framework is intended to provide communications between multiple heterogeneous schedulers whether they operate at the local or cluster level.

Lauret et al. from the Brookhaven National Laboratory discuss the STAR scheduler [44]. The STAR is a Meta-Scheduler which allows users to submit a job on several input files by dividing the job into different processes that are dispatched to different machines. It acts as a wrapper on the current STAR infrastructure to interface it with Grid middleware and takes into account preliminary network information when taking decisions. WP1 in the Data Grid Project (now EGEE) created a resource broker (which was interfaced to ROS as described in [31]) under the EDG workload management system which is an extended and derived version of Condor and is subject to the same issues and problems as Condor. Although the problem of bulk scheduling has begun to be addressed in the most recent version of gLite (through the idea of so-called shared “sandboxes”) the approach taken does not address data intensive scheduling. SPHINX [46] is a framework for workflow management and execution on heterogeneous platforms and is a data-intensive scheduling engine. Initially it is being used

for scheduling data-intensive applications on Grid but it is very immature, under development and does not support bulk scheduling.

#### **2.4.2 Application Level Scheduling**

Fast networks have made it possible to coordinate distributed CPU, memory, and storage resources to provide the potential for application performance superior to that achievable from any single system. Each Grid application is scheduled by its own AppLes (Application Level scheduling) in the AppLes project [47] and the key to this approach is that everything in the system is evaluated in terms of its impact on the application. There is a difference between a job and an application. A job is the work being accomplished within a single unit of work whereas the application is the collective work among all the jobs that comprise the application. Applications features and the Grid status have a strong impact on the system performance and on the accuracy with which the user forecasts. It is not designed to be used for data intensive and network aware applications and mostly takes cycle consumption as a decision criterion. Elmroth and Peterg [48] describe a Grid wide fair share scheduling system for local and global policies. They feature quota based scheduling and multilevel queues, although they do not consider reprioritisation and it was not P2P oriented. The GridWay Scheduler [70] provides dynamic scheduling and opportunistic migration but its information collection and propagation mechanism is not robust and in addition it has not as yet been exposed to bulk scheduling of jobs. Jin et al. [49] describe an adaptive Meta-Scheduler that considers availability of the computational, storage and network resources in the scheduling decisions. Although our approach is closer to this one, we focus on job execution on multiple sites as well as consider dynamic network characteristics as prime decision criteria.

#### **2.4.3 Economy Aware Scheduling**

The geographic distribution of resources owned by different organizations with different usage policies, cost models and varying load and availability patterns can be problematic. The producers (resource owners) and consumers (resource users) have different goals, objectives, strategies, and requirements. To address these resource management challenges, Buyya et al. has created a distributed computational economy-based framework Nimrod-G which uses Grid economies to implement its scheduling policies. It is good for economy-aware applications but not for bulk scheduling and network-aware applications.

## 2.5 Batch Scheduling and Execution Systems

In this section selected batch scheduling systems are discussed which have made notable advances in the domain of this thesis research. Batch systems are the early entrant in the scheduling arena and due to their maturity and wide scale usage most of these systems (except Condor) have been launched commercially. Some companies have started marketing their products whereas others are still in development. Most of these were started as research projects but due to their popularity, later became part of the commercial domain under the umbrella of their off-shoot companies.

Basney and Livny from the University of Wisconsin describe the advances made in the Condor project. Condor is a high-throughput computing environment and can manage a large collection of computers such as PCs, workstations, and clusters that are owned by different individuals. It offers powerful and flexible resource management services for sequential and parallel applications. The Condor system has been enhanced to support the creation of personal Condor pools and a data scheduler for data-intensive scheduling. However its data and compute services do not work coherently and seldom employ network characteristics to make scheduling decisions.

Gribble et al. describe the LSF project from Platform computing [60]. LSF is software for managing and accelerating batch workload processing for compute and data-intensive applications. With Platform LSF, one can intelligently schedule and guarantee completion of batch jobs and can organize individual jobs into a group for greater control and management. It can also provide Meta-scheduling capability in association with the Community Scheduler Framework (CSF) [43] and the Platform Globus Toolkit. LSF is a local resource management system and CSF does not consider network traffic in queue management and Meta scheduling decisions. Moreover, Meta scheduling is strongly coupled with Globus and has all the inherent problems which Globus is facing. Moab Grid Suite has developed Silver [90]. The Silver project is designed to be an enterprise level Grid scheduler and takes advantage of the capabilities found within the Maui Scheduler to provide load balancing, co-allocation, data scheduling and quality of service guarantees at the Grid level.

Chapin et al. from University of Virginia describe the Legion [62] project. Legion is an object-based Grid operating system which provides the software infrastructure so that a

system of heterogeneous, geographically distributed, high performance machines can seamlessly interact. It provides resource reservation capability and the ability for application level schedulers to perform periodic or batch scheduling. The Grid Engine project from Sun Microsystems is a Distributed Resource Management (DRM) system to aggregate compute power and delivers it as a network service. Both Legion and the Sun Grid Engine provide a local execution and local scheduling environment and have no mechanism for meta scheduling, a fundamental requirement for DIANA scheduling.

## **2.6 Meta Scheduling Algorithms**

Efficient scheduling across Grid nodes is necessary to maximize application performance, regardless of the efficiency of the scheduling algorithms. The important aspect of scheduling is policy enforcement and resource optimization and this is achieved through selection of the best resources to run the jobs. The brain of any Grid scheduling system is the underlying algorithm which is being employed to select the resources for efficient job execution. Different algorithms use different scheduling approaches. Given below is a detailed survey and critical analysis of the related scheduling algorithms.

### **2.6.1 Traditional Scheduling Algorithms**

Greedy algorithms always take the best immediate, or local, solution to finding an answer. Greedy algorithms find the overall, or globally, optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems. If there is no greedy algorithm that always finds the optimal solution for a problem, one may have to search (exponentially) many possible solutions to find the optimum. Greedy algorithms are usually quicker, since they don't consider the details of possible alternatives but may not be suitable for bulk scheduling when large number of jobs are in the queue.

Altinbukan et al. state that the fair-share scheduling [98] is a scheduling strategy for computer operating systems in which the CPU usage is equally distributed among system users or groups, as opposed to equal distribution among processes. Although it provides good Quality of Service (QoS) features it is insufficient in functionality if we want to include other than compute resources. UNIX is fundamentally a time-share operating system that employs a round-robin scheduling algorithm. Each process is placed in a run-queue and allocated a

service quantum of time. Round-robin favours short process demands. The difference between the round-robin and FIFO (First In First Out) [99] scheduling algorithms is simple. FIFO will allow a process to run indefinitely which can create starvation for some Grid jobs. Round-robin will force a process to yield and allow another process of equal or higher priority to run if it is able. Both FIFO and round-robin are not suitable for DIANA scheduling; we require an adaptive scheduling mechanism which can make dynamic scheduling decisions based on the data locations, network conditions and site loads.

### **2.6.2 Scheduling Optimization Algorithms**

Greedy algorithms try to schedule jobs in real time and often ignore optimization preferences. On the other hand, scheduling optimization algorithms focus on the scheduling efficiency and optimal job execution. Backfill [64] is one such scheduling optimization that is based on earliest-job-start information. Feasible backfill jobs are filtered selecting those that actually fit the current backfill window and the job with the best fit is started and the backfill window size is adjusted accordingly. Priority queues, as explained by Hubertus Franke et al. [65], are a fundamental class of data structures being used in the design of scheduling algorithms for a long time. It is an abstract data type to efficiently find the item with the highest priority across a series of options. Their uses range from the application level scheduling to the lowest levels of the operating system kernel.

Yarkhan and Dongarra describe the Tabu Search [86] algorithm for scheduling optimization. In a Tabu Search, some historical information related to the evolution of the search is retained (basically the itinerary through the solutions visited) to improve the efficiency of the exploration and scheduling process, Such information will be used to guide the movement from one solution to the next one thereby avoiding cycling. Takefusa et al. propose Bricks [66] which is a performance evaluation system for scheduling algorithms on the Grid and investigates and compares the performance of different data Grid models. Bricks can simulate various behaviours of global computing systems, especially the behaviour of networks and resource scheduling algorithms. But this is only a performance evaluation environment which can facilitate to test our own algorithms and does not provide a meta scheduling solution.

### **2.6.3 Global Cooperative Scheduling Algorithms**

In large scale distributed systems, jobs will compete for the scarce resources and this is likely to distribute the load disproportionately among the Grid nodes. Therefore such scheduling algorithms are required which allocate the best available resources to a job but only after checking the state of all the jobs and resources in the Grid so that the throughput of the whole Grid is maximized rather than improved throughput for few users or jobs.

Lewis and Paechter discuss simulated annealing (SA) [67] as a scheduling optimization approach. It is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. Simulated annealing is a generalization of the Monte Carlo method used for optimization of multi-variable problems. Possible solutions are generated randomly and then accepted or discarded based on the difference in their benefit in comparison to a currently selected solution. Patrick and Piotr discuss the scheduling applications of Game Theory [68]. Grid scheduling can also be modelled using game theory, a technique commonly used to solve economic problems. In game theory each of a number of players attempts to optimize their own payoff by selecting one of many strategies. Vincenzo et al. state that the continuing price/performance improvements of computational systems have made the Genetic algorithms [91] attractive for some types of optimization. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive. Aida et al. discuss the Branch-and-Bound Algorithms [92]. These are the counterpart of the backtracking search algorithm and the algorithm traverses a spanning tree of the solution space using the breadth-first approach. A queue is used, and the nodes are processed in first-in-first-out order. If a cost criteria is available, the node to be expanded next (i.e., the branch) is the one with the best cost within the queue. This is useful only when there are limited numbers of jobs to be scheduled but for a higher number of jobs only the least cost jobs will be placed, which is not a very optimal approach.

### **2.6.4 Priority Based Scheduling Algorithms**

A common type of scheduling algorithm which can meet the different Quality of Service (QoS) requirements is priority-based scheduling. The idea is to rank processes based on their priority and their need for processor time. Processes with a higher priority will run before

those with a lower priority, while processes with the same priority are scheduled round-robin (one after the next, repeating). Noriyuki et al. discuss the Suffrage-C [69] algorithm. It is a revised version of Suffrage [100] so as to make it easier to implement. Suffrage-C gives each task its priority according to its suffrage value. For each task, its suffrage value is defined as the difference between its best completion time and its second best completion time. The suffrage value of each task varies over time because of the change of processor speed in a Grid. The idea behind Suffrage-C is that a processor is assigned to a task that would suffer the most if that processor would not be assigned that task. Min-min and Max-min [93] are based on static algorithms. Min-min gives the highest priority to the task which can be completed earliest. The ties are broken arbitrarily. The idea behind Min-min is assigning tasks to processors that will be executed fastest. Max-min gives the highest priority to the task with the maximum earliest completion time. Graham et al. describe WQ (Work Queue) algorithm [94]. It is a classic algorithm that was originally developed for homogeneous parallel machines. WQ arbitrarily gives priorities to tasks and arbitrarily breaks the ties. The idea behind WQ is that faster processors will be allocated more tasks than slower processors.

## **2.7 Meta Scheduling and Grid Standards**

Grid components in general and scheduling in particular need to possess a high degree of interoperability and facilitate robust communication to adjust to ever-changing needs and requirements. For this to happen, we need standards for designing a complete Grid architecture as well as for scheduling and resource management so that common interfaces and common solution to various problems can be found. This is the task of the Global Grid Forum (GGF) and GGF scheduling effort was discussed in section 1. Other organizations which are important to GGF are: IETF for Internet standards [71], DMTF for distributed management standards [72] OASIS for e-business standards [73], WS-I for Web services interoperability, and W3C for interoperability for the Web [74]. These standard bodies are not only working for common interfaces, they are also proposing very powerful solutions to some challenging problems. Presented below are brief descriptions of only those standards in Grid scheduling which are relevant to the current area of research. There is no single scheduling standard that can adequately address the DIANA scheduling requirements.

### **2.7.1 Scheduling Architecture**

Grids will provide a large variety of compute and data intensive services. The interactions of those services require extensible and integrated resource management. Such a coordinated scheduling of services is currently not available. However, such Grid scheduling is essential for most applications. The GGF Grid scheduling Architecture defines a scheduling architecture that supports cooperation between different scheduling instances for arbitrary Grid resources. Grid scheduling optimization techniques and Grid Economic brokering are the core activities being pursued by the group nowadays. The complexity of resource management and scheduling tasks increases as the number and types of resources requiring management increases, and is further complicated when those resources are distributed as on the Grid. This issue is being addressed through the development of manageability standards in the Open Grid Services Architecture. WSRF [76] is a collection of specifications to support Grid services or other stateful resources. Normally, Grid services are assumed to represent some resources that have a state. WSRF suggests how to manage the context in the case of a stateful resource. These stateful services are indeed a very powerful feature and can increase QOS in scheduling and resource management and therefore can be useful in the DIANA scheduling approach.

### **2.7.2 Scheduling and Resource Management**

Scheduling and Management are interrelated in the Grid and scheduling is in essence the optimized management of Grid resources and execution services. The Scheduler is a component of the Grid resource management system and takes care of all the scheduling policies. The Distributed Management Task [72] Force (DMTF) and its Utility Computing Working Group (WG) is working to “Unify the computer industry on a common manageability model and profiles for utility computing” under Web Services Distributed Management [85]. Scheduling is also offered as a service and this standard can optimize the scheduling process effectively. The GGF Distributed Resource Management describes the Distributed Resource Management Application [77] API (DRMAA) and its scope is limited to job submission, job monitoring and control, and retrieval of the finished job status. This standard can optimize the Meta scheduling process since all local schedulers can interact with each other using this standard. Web-Based Enterprise Management (WBEM) [78] is a set of

management and Internet standard technologies developed to unify the management of enterprise computing environments. UPnP [79] can be used to manage services and resources at lower level whereas WBEM can facilitate Grid scheduling in enterprise systems.

The GGF CDDLM (Configuration Description, Deployment, and Lifecycle Management) [80] Group has created specifications to describe the configuration of services to deploy them on the Grid and to manage their deployment lifecycle (instantiate, initiate, start, stop, restart, etc.). Efficient configuration and lifecycle management of the services can have direct impact on the service availability, access and execution and this can add a good amount of performance in the scheduling decisions. A reservation is a promise from the system that an application will receive a certain level of service from a resource and this is managed by the GGF Standard on Advance Reservation [81]. It allows users to make reservations in advance of when the resource is needed and is capable of making and manipulating a reservation regardless of the type of the underlying resource. Reservations are expensive operations from the resource management point of view and we will not follow this mechanism in the DIANA scheduling to provide the QOS, rather priority driven approaches are more optimal in global cooperative scheduling.

### **2.7.3 Scheduling and Monitoring**

System and network monitoring have gained key importance in modern day distributed systems and are used to collaborate and cooperate in performing a wide range of information gathering and processing tasks which are the backbone for efficient scheduling decisions. Network monitoring systems can analyze and process the information, in a distributed way, to provide optimized scheduling decisions in large scale distributed applications. The Grid Monitoring Architecture standard describes the major components of a Grid monitoring architecture [82] and their essential interactions. The Simple Network Management Protocol (SNMP) [83] can be used to monitor network-attached devices for any conditions that warrant administrative attention. The Grid High-Performance Network (GHPN) standard [84] deals with congestion control, large network optimization, and network managed services which are crucial to optimized scheduling.

## 2.8 Conclusions

This chapter described the background of this thesis's research agenda and what is the state-of-the-art in the Data Intensive and Network Aware scheduling domain. Related research was presented and an analysis of the current and previous efforts was discussed. Research papers, standards and projects which have a direct contribution or can help to diagnose the problem and find solutions or that can become an alternative approach have also been discussed. In conclusion, there are three major resources in this type of scheduling commonly known as data, network and computation, and scheduling optimization is not possible until all these resources are managed in the scheduling system. Furthermore, bulk scheduling has not been addressed in the existing literature and there are no adequate solutions in handling the number of jobs submitted in bulk scheduling and the associated issues with the quality of service which arise in such scheduling. Consequently this chapter has established the state-of-the-art in the domain of this thesis's research problem defined in chapter 1.

This chapter has helped to identify the issues and possible approaches which are being followed by others and what is missing and needs to be provided to address this core issue of Data Intensive and Network Aware scheduling. The following chapter identifies the requirements for a Data Intensive and Network Aware scheduling problem. The requirements identified in chapter 3 elicit and illustrate the hypothesis and research questions described in chapter 1. The hypothesis, research questions and the emerging requirements will become the heart of this thesis and subsequent chapters will address these requirements.

## **Chapter 3**

### **Data Intensive and Network Aware (DIANA) Grid Scheduling Requirements**

The previous chapter provided an in depth overview of the state of the art about DIANA scheduling in the Grid environment. Accomplishments which have been achieved in current and previous research were discussed in detail. A critical analysis of the existing literature was provided to help identify the missing points where further research is required. It was ascertained that DIANA scheduling is not adequately served by the existing research and that no existing scheduler and scheduling algorithm can co-relate huge amounts of data and their location(s) with computation and network capabilities during the scheduling decisions. It was pointed out that no current scheduling system takes scheduling decisions by taking all three parameters into account. In chapter 3, a detailed report of the research questions as presented in chapter 1 is provided in the form of a requirement analysis for the DIANA Grid scheduling. These research questions are divided into small problems and presented as requirements in this chapter.

In this chapter, justification is given for selecting the Use Case Model as a requirement analysis tool and its salient features are highlighted very briefly in section 3.2. Next the DIANA scheduling requirements have been extracted from the High Energy Physics (HEP) use case as they appear and are described in the requirements section 3.3. In Section 3.4, an analysis of these requirements is presented and all major requirements are classified as compute, data and network related requirements which will be followed in the later chapters of thesis. By compiling the use cases and requirements, this chapter attempts to illustrate all the relevant usage scenarios in a succinct way. This chapter is concluded with a concrete set of requirements which will be incorporated in the proposed DIANA scheduling approach.

#### **3.1 Introduction**

This thesis follows an experiment driven and prototyping approach. In this model we build something and then that artefact is used as the basis for a more generic class of solutions.

There are many reasons why such an approach, however, can be unsatisfactory for research. The main objection is that it carries considerable risk. For example, the artefact may fail long before we learn anything about the conclusion that we are seeking to support. Indeed, it is often the case that this approach ignores the formation of any clear hypothesis or conclusion until after the artefact is built. This may lead the artefact to become more important to the researcher than the ideas that it is intended to establish. This “proof by demonstration” approach has much in common with current rapid prototyping practice. Iterative refinement can be used to move a prototype gradually towards some desired solution; the evidence elicited during early prototypes can be used to better define the goal of the research as the work progresses. This approach has however two main limitations. Firstly if the system fails then one may have gained few insights into the basic research question [101]. The failure may be more due to the limitations of the implementation than to the idea itself or due to insufficient requirements. Secondly it can be difficult to generalize from a specific system to generic solutions.

While the second reason for failure can be avoided by suitably crafting the research problem, the former issue can have serious consequences if a remedy is not found at the outset of the research. In order to address this potential problem a Use Case Modelling approach is followed for the requirement engineering phase of DIANA scheduling to ensure that the correct assumptions have been made in the initial project phases. Waterfall models [102] [103] of system implementation have been demonstrated to be less successful in satisfying user requirements by adopting designs which may not necessarily address all the aspects of user involvement at the early stages of system analysis and design [104]. The Use Case approach is a very flexible technique due to its iterative and controlled nature. It aims not to “fight change” but rather to embrace change as a core systems implementation driver.

Grid Computing and consequently Grid scheduling follows the Service Oriented Architecture (SOA) paradigm [105]. Each component behaves as a service, performs autonomously and has a self-contained behaviour. These characteristics demand an approach which is iterative, adaptive and descriptive so that each component or service can be modelled in a self-contained and autonomous manner, but at the same time all services should work cohesively and homogeneously in its entirety. By requirement engineering, we are in fact eliciting and explaining the research questions. We do not know the solutions to these research questions

or requirements until we investigate them, but by dividing the research questions into smaller approachable problems and iterating through the requirements process, we can approach the research problems in a better and organized manner. In the following sections different aspects of the system are illustrated from a requirements point of view and various features of this approach are applied to capture requirements.

### **3.2 Requirements Engineering Process for DIANA Grid Scheduling**

Requirements' gathering is the first phase within the research and development of a software system. The various subsequent project phases are based on the information gathered and documented within requirements engineering, therefore the output of the requirements engineering phase (the requirements model and the specification document) are crucial to project success. The specification document is the basis for communication and reporting and all future aspects of the developed system and its functionality have to be clarified using this document. Unfortunately even then there may still be serious problems that arise in research and development of software systems. According to an ESPITI survey [119], many software projects fail, are far over time schedule or cost significantly more than expected. The roots of the problem often originate in the requirements engineering activities. There may be serious problems in the research and development phases due to the poor, misunderstood or even missing requirements.

In order to avoid these problems, a multi-prong strategy has been adopted to collect the requirements for the Data Intensive and Network Aware (DIANA) Grid scheduling. We surveyed the literature thoroughly as detailed in Chapter 2 to explore the state of the art related to data intensive scheduling. Consideration was given so that no "wheel re-invention" was encountered and where possible re-use of existing documented requirements from all the major projects such as EGEE and Open Science Grid etc was followed and related material was refined according to the research problem needs. A set of requirements emerged out of these studies which were cross examined to eliminate shortcomings.

A Use Case Model was adopted to elicit and capture requirements since this was an easy way to get comments from experts and to illustrate the requirements in an easy-to-understand manner. The Use Case Model is a powerful tool for controlling scope throughout a project's life cycle. Since a simplified Use Case Model can be understood by all project participants, it

can also serve as a framework for ongoing collaboration as well as a visual map of all agreed-upon functionality. It is, therefore, a precious reference during later negotiations that might affect the project's scope. A key point of our strategy in requirement engineering is that we should be concerned about "What" and not about "How", since requirements specify "what" shall be provided and not "how" -- the how is a design aspect rather than a requirement.

Documenting the rationale for each requirement (i.e. why it is required) is a good technique to reduce the number of requirements [120]. This gives the flexibility of implementing the system by harnessing the available resources and leaving the rest of the detail to others without influencing their choice of technology and the implementation details that may later follow. This approach can also vary from team to team keeping in mind the diversity of the expertise in a particular domain and the background of the team or researcher. Following this philosophy also endorses the iterative approach which we will be following during the complete research process presented in this thesis. Hence each component and module is self contained, has specific deliverables and requirements are tailored to fit in this principle. This increased fault tolerance and communication will reduce tight coupling in case more than one team works in parallel when the research prototype moves into a production phase.

### **3.3 Requirements Extraction from the Use Case Model**

A High Energy Physics (HEP) use case is discussed to provide an overall view of the system and provide details of the various components and how they are likely to interact with the system. The research problem that is tackled in this thesis is explained through this use case and divided into small realisable requirements. Based on the use case, the following sections summarize the requirements. These requirements will be better understood if the context and the target environment are presented under which these requirements emerge. It is important to mention here that DIANA scheduling has most of its applications in the data intensive analysis domain such as HEP analysis and therefore, we illustrate here a comprehensive Grid analysis scenario and its use case for the HEP environment in order to capture the requirements as they appear. As the use case is being discussed, at each step in the use case we identify a number of features which are required to carry out such an analysis and these are presented as specific requirements.

Large-scale data-intensive problems, such as those that arise in the HEP experiments currently being developed at CERN, can generate petabytes of scientific data. Several hundred end-users can run analysis jobs at the same time, processing large amounts of this data (up to several hundred TB) replicated over several tens of Grid sites. Used in this way, the Grid will become a highly dynamic, large-scale environment with unpredictable access patterns as described by Rang Nathan et al. [17]. The data-intensive jobs will need to take data location into account when scheduling the jobs on the Grid resources. The replication of data can help improve the scheduling and execution optimization by reducing the frequency of remote data access. Moreover, large numbers of jobs and resources can make the centralized algorithms ineffective. Both replica management and computational job placement need to be optimized to make best use of all their resources. Initially the cost for transferring a file between two services should be sufficient, and this cost should be the predicted time it takes for that file to be transferred. Furthermore, network services should provide a useful cost that can be used to optimize their use. Normal network use should also be given a cost for optimization purposes.

**Requirement#1:** Data Location should be taken into account when making scheduling decisions. Delay and bandwidth of a site are the factors that influence the choice of data location.

**Requirement#2:** The best replica of a dataset should be considered while scheduling Data Intensive Jobs. This can reduce data transfer time.

**Requirement#3:** Decentralized systems should be employed for bulk scheduling of jobs. The current client-server model of Grid scheduling systems is not scalable and cannot cope with the job frequency of bulk scheduling.

**Requirement#4:** Those sites should be given priority in replica selections which have better and stable networks. (Stable here implies a reliable network which does provide a comparatively better quality of service).

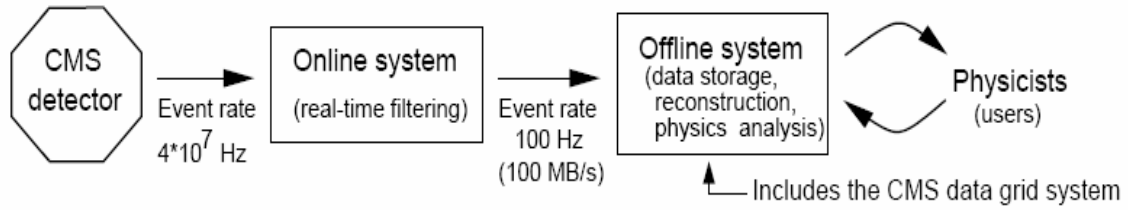


Figure 3.1: The CMS online and offline systems and their environment.

For example, in the CMS experiment at CERN, CMS Production and CMS Data Analysis are two very data and computation intensive processes [139]. CMS Production [109] covers three areas: the generation and simulation of data (cmsim), the Hit Formatting, and the Digitization and pileup. As shown in the Figure 3.1, CMS Data Analysis [110] deals with the issues of so-called event reconstruction, the selection of the physics events and the visualization of the data through a visualization tool such as ROOT, IGUANA, JAS, WIRED etc. In both cases, physicists submit, individually and collectively, millions of jobs and this is known as bulk submission in which each job accesses some subset of that data in the Grid.

**Requirement#5:** Scheduler should have a special mechanism to manage bulk scheduling and execution.

The resource access patterns used in physics analysis (see Figure 3.2) tend to be less predictable [112]. This comes from the fact that jobs are initiated from almost any HEP site in the world and data is not uniformly distributed across the Grid. Some sites have the required datasets replicated to them whereas others sites do not have access to the datasets. The replication and availability of the datasets influences the job submission and execution patterns. As shown in Figure 3.2, a physicist runs analysis jobs. He may either execute an inclusive analysis, using all the collected data, or select interesting events using Tags. A set of events containing similar physics is sometime called a “channel”. Channels of interest are analysed starting from Analysis Object Data (AOD), accessing parts of the Event Summary Data (ESD), or even of the raw data, if necessary. The need to access different portions of the data increases sparseness. The generated data may be private to the physicists, possibly with links to full events or other objects located in the official datasets. These data can be stored on private storage or they can be registered on the Grid in a private area accessible to the owner. Systematic effects are studied by looking at the ESD for small event samples. Access

to complete individual events ( $\sim 100$ ) may be required and these are studied in detail e.g. with an event display. A typical job will perform some calculation on a specified input dataset and will produce some output. It can be interactive or batch and is part of the dataflow explained above. There are two main cases of HEP jobs: organized jobs and chaotic jobs.

Organized jobs are planned in advance and perform a homogenous set of tasks. The input is a pre-determined set of events accessed sequentially, processed and then written out, in a different format, suitable for calculations to be performed in a subsequent phase. A production team manages the data processing; simultaneous requests to the same input dataset are minimised by a proper organisation of the production. Each job needs to have a cost based on the amount of resources used (CPU, memory, storage, bandwidth) and on the priority with which these resources are used. Furthermore the Scheduler must be able to calculate the job cost on the basis of the information provided by the user and that it should use this information to schedule and optimize the job execution. Users need to know the cost to access a physical copy of a dataset on a specific Storage Element (SE) across the Grid environment (including the cost of putting the copy there if one is not already available there). That dataset should be selected which has the least data transfer cost. In choosing the best access to the dataset, it is required for the scheduling system to have considered the cost options (respecting any possible protocol constraints).

**Requirement#6:** Jobs should be assigned to resources that possess the required data set provided it is cost effective.

**Requirement#7:** Data and Jobs should be moved to a third location if feasible for scheduling and execution efficiency.

**Requirement#8:** The cost of computation, data transfer and network should be taken into account before making a job or data movement decision. This requirement clearly relates to the research hypothesis of this thesis as do the other requirements. Any requirement that emerges from the use case but is not related to the hypothesis will not be addressed in the coming chapters.

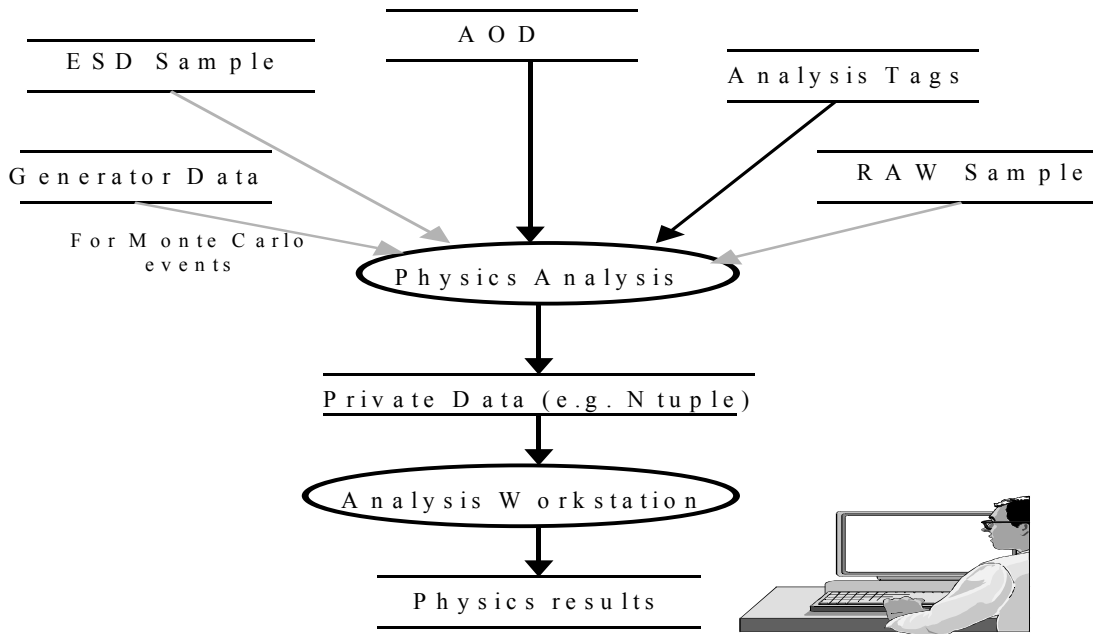


Figure 3.2: User analysis

Chaotic jobs are submitted by many users acting more or less independently, and encompass a wide variety of tasks. The input is typically a selection/analysis algorithm to be applied to a very large dataset. Users can submit jobs of this kind at any time, simultaneously asking for the common input datasets. A user may want a shorter time for a solution for a given job than the default one. This can translate in a higher placement of the job in the batch queue of the Computing Element (CE), so that it enters execution more rapidly, and a higher execution priority on the Worker Node(WN), so that it gets a larger share of the WN CPU. Such a system will also need associated controls on priorities. For example, the system might deduct a larger amount from a user's resource usage quota for a high-priority job than it would for the same job submitted with normal priority. Another possibility is to limit high-priority jobs to a special class of users. The system also will need to deal with priorities that differ from site to site. There must be the ability to request for a scheduled reservation of resources, i.e. with both the beginning and termination of the resource's reservation and thus availability in a future timescale. Although reservation is one of the requirements in supporting data analysis, there is little provision in the Grid middleware to support network reservation. Efforts are underway in major Grid projects such as EGEE to address the resource reservation issue.

It is common for applications to know in advance about a future need for transmission of a certain amount of data. In those cases where it is crucial that network resources are available at the time when the data is ready for transmission, it is also essential to have reserved the network resources beforehand so as to ensure timely delivery of the data at the destination. The Scheduler can also be asked to execute a job and produce a dataset (DS) within a predefined time (see Figure 3.3 [111]). Only high priority jobs can be given the privilege to run on dedicated or reserved resources to complete the job in a limited amount of time. This option should be used only in case of extreme circumstances and suitable scheduling policies are required to accommodate such jobs.

**Requirement#9:** There should be a mechanism to prioritize the jobs for providing quality of service to certain high priority jobs.

**Requirement#10:** Resource Reservation should be provided if we know in advance when the users plan to submit high priority Jobs. This requirement has little relevance with the thesis hypothesis and will be excluded in the analysis of requirements in the following sections.

**Requirement#11:** A Scheduler should support time constrained mechanism to execute the jobs within the user specified time limit. Time constrained and deadline scheduling is not real time scheduling which is used in the multimedia applications.

Since analysis jobs are not the same as production jobs, the same scheduling model cannot necessarily be applied to each. The fundamental difference is in the input data. It is very unlikely that in the case of the analysis jobs all bulk submission will be using the same input data. On the other hand this can be true in the case of computation intensive production jobs since the nature of all the jobs is similar and jobs are not dependent upon any input data transfer, rather they produce only output data. If bulk submission is taking different datasets then it is likely to overload the site as well as the network. This also discourages data transfer and forces job submission to a site where the data resides, which is not a very efficient scheduling approach. This scenario occurs if certain datasets are used for a whole group of researchers. For instance, in high-energy physics several very large data sets are used by a large research community. The transmission of such data to a specific resource in order to start a single job is often not efficient as it would require a large amount of storage and a long time for the network transfer. Instead, jobs may be assigned to resources that possess the

required data set. Nevertheless, it may be justified to replicate the data to a specific site if enough jobs would benefit from that resource.

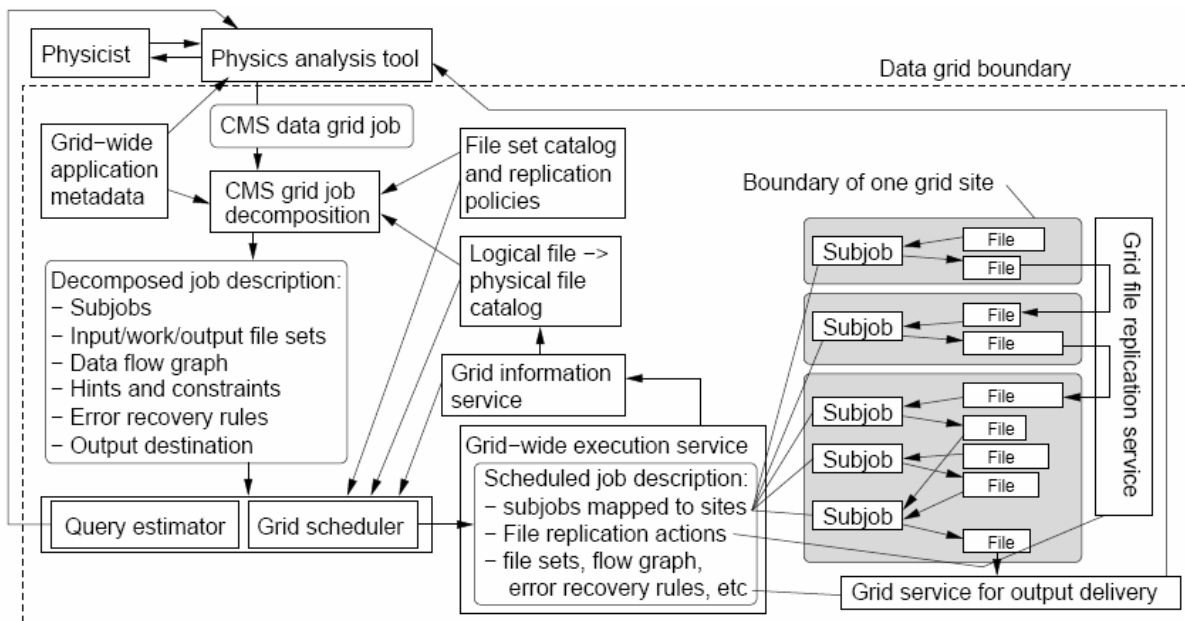


Figure 3.3: Creation and processing of CMS data Grid jobs

Users need to access remote portions of a dataset. In HEP, for example, datasets contain one or more so-called “events”. Some jobs will read only a few events per file. The data of each event is stored as a set of ‘data products’. The term data product is used for a small self-contained unit of data. In CMS terminology, data products are usually called ‘objects’. Data product values are always stored in files and normally there are many related data product values in a single file. The related files are stored in a single dataset and in the CMS environment only datasets are replicated and used by the jobs. If the fraction of data is small enough (30GB), jobs could execute more quickly if they could access single events rather than accessing entire datasets or making local replicas. If the middleware can provide this service, it may need a hint to indicate that a given dataset may be opened remotely. The trade-off is between complexities of programming (providing single-event access) vs. typical event sizes and typical bandwidth. Given sufficient bandwidth and small single events, it might not be worth retrieving a portion of a dataset and in this case, this is no different than DS access. When the middleware has to schedule a job and transfer a large amount of data, it may have several alternatives from where it could retrieve the data or to where it could send the data. Predicted available bandwidth would allow the middleware to make the best choice

concerning the network resource to use (the path to use). The prediction of available bandwidth would also be useful to answer the job request which requires transferring some data within a time window.

**Requirement#12:** All jobs should be scheduled to a site which has the required dataset and being accessed by the jobs.

**Requirement#13:** It may be required for a job to retrieve the portions of data from a remote dataset.

**Requirement#14:** Scheduler should find a best network path based on the latest site-site calculations between storage and computing element.

**Requirements#15:** Parameterized or specialized schedulers are required to optimize the scheduling process. These schedulers should be tuned for specific type of applications and for certain parameters which are of special interest for the applications.

Bulk submission is derived from a batch type of execution in which a batch of jobs is submitted and users must wait until the final result is produced. However, analysis jobs are more interactive in nature and each job has different user priorities. Consequently, users should not be forced to make their jobs part of the bulk submission until it is cost effective. A more feasible approach is that a Meta-Scheduler could calculate the cost (i.e. the computation, data transfer and network cost) of each job at a global level and then the job could be submitted to a site which has the least cost. This will reduce queue time, processing time and will help to organize the Grid. The Scheduler needs to monitor job progress and access the output of the running job. It will help to perform management or control functions on a job. The Scheduler should send jobs to specific sites for optimal utilization of the resources. This step is needed to provide dynamic and adaptive resource management due to the volatile nature of the Grid environment. This is a step forward to making Grid a real time system and to make intelligent decisions strategically in a consistent way, taking into account the changing state of the system.

**Requirement#16:** Scheduler should have the capability to steer and move the jobs to the sites having better resources.

**Requirement#17:** If Job is moved to a remote site, scheduler should ensure that requirements software is available for the job execution.

**Requirement#18:** The scheduling system should calculate and incorporate the network measurements before planning a job submission.

### **3.4 Analysis of the Requirements**

While explaining the HEP analysis use case in section 3.3, requirements were captured as they appeared. Now we present here the brief analysis of the major requirements for the DIANA scheduling process. Indeed, this section expands and explains the particular requirements related to DIANA scheduling which were discussed in the above mentioned HEP use case. There are at least three major requirements areas for the Grid Meta-scheduling System that have a direct bearing on the performance needs of applications and these are described here as compute-related, data-related, and network-related requirements.

#### **3.4.1 Compute-Related Requirements**

Compute-intensive applications drive the need for scheduling and resource management services that can quickly and optimally locate high-performance computational resources given the particular requirements of those applications as well as those of the end-users. In locating and scheduling jobs on those resources, resource management services need to take into account not only the compute related needs of an application [113] such as load, available computing capacity, data location etc. (see requirement #1) but also key factors [17] such as the time at which the application is required to run, the cost and availability of the resource (see requirement #8), and the efficiency of application on that particular resource. There are many other factors that may also be taken into account, such as the consistency of the resource like machine failure rate (see requirement #16), the network characteristics for transferring results to mass storage (as described in section 3.4.2) or the user quota and privileges on the intended resource as well as any firewall security policies that could prevent the use of remote resources. Since users can submit any number of jobs at any time, there should be a mechanism to prioritize the jobs for providing quality of service to certain high priority of jobs (see requirement #9). The scheduler should self-organize themselves by exporting jobs to the least loaded sites in case of job loads as may be the case in bulk

scheduling and scheduler failures (See requirement #19). This will not only save the jobs from starvation but will also help the scheduling optimization. Scheduler should have the capability to steer and move the jobs to the sites having better resources (see requirement #16).

Grids present a challenge for applications in that they manage and share heterogeneous resources, so may consist of many different types of computing platforms and operating systems (and versions of operating systems). Resources also have varying software installations and execution environments, such as shells, compilers and libraries. A given application may be compatible with only a limited set of computing architectures and software. Even when an application is portable across many platforms its particular characteristics may dictate that it will run more efficiently on a given architecture. Jobs should not be submitted to sites which do not have required execution libraries or software available since this may result in job failures. These needs must not only be taken into account by the scheduling services, but also dictate the need for applications to be able to sufficiently describe their requirements.

**Requirement#19:** The scheduler should be scaleable and self-organizing to cope with bulk Job scheduling. Here self-organizing behaviour implies the capability to export jobs to least loaded resources or to import jobs from over-utilized sites.

**Requirement#20:** Scheduler should schedule jobs to only those resources where required execution software (particular application specific libraries) is available.

**Requirement#21:** Scheduler should also take into account the software and heterogeneity into account when scheduling jobs to remote sites.

### **3.4.2 Data-Related Requirements**

Most applications have at least some basic file management and data placement needs [17] that place constraints on how the computational resources are scheduled in a Grid environment. Applications operate on a set of input datasets and produce output datasets to be analyzed or processed by a second application to visualize the resulting data. The size of the input and output datasets and the amount of overall storage space allotted to a user

necessarily has some bearing on the scheduling of such applications [123]. If the input datasets must be retrieved from a remote location, then the time required to transfer the datasets must be taken into consideration when scheduling compute resources for the given application (see requirement #13).

There are a number of methodological issues that impact on scheduling performance. One such approach is data-aware scheduling, the process of scheduling the tasks closer to the data that they require [115] [19] (see requirement#6). Firstly, considerable performance improvements can be gained by taking into account the amount (and speed) of data transfer that is required during a task execution [117]. Secondly, it must be possible to access the contents of files efficiently according to the data model since datasets may be distributed among several database management systems. Thirdly, there should be provision in the scheduler to move the data towards the computation site if considerable performance gains are guaranteed by this data transfer and placement and this is becoming a reality due to the high available bandwidth in modern optical networks (see requirement#7). This is discussed further in section 3.4.3. Location of the data is a key to the Data Intensive scheduling and scheduling optimization is not possible without knowing the location and then selecting a best place to fetch data (see requirement#1). Data Size, location and possible decision to replicate the particular dataset can have significant impact on the scheduling optimization (see requirement #2). Similarly, Data transfer and access costs are other important data related parameters which should be incorporated in the Data Intensive scheduling process. Some times this process also involves replication and the jobs then use this new dataset when it is scheduled on that particular site to improve the scheduling optimization. Further all the jobs in a job burst can be scheduled on a single site if it is cost effective in terms of execution and queue times (see requirement#5).

Data Intensive jobs read and store their data on some tertiary storage system [118]. Access latencies of such tertiary systems can be of the order of seconds up to hours [119] in case the data resides on a tape that is not mounted yet. With the recent trends in the network performance like Bandwidth challenge of 131 Gbs and even more in years to come, it might be more cost effective for the data intensive jobs to access and transfer the data over the internet from the remote locations than accessing the tertiary storage in the local environment

(see requirement#12). We should also consider the tape access and associated latencies when scheduling the data intensive jobs since they can be much larger than the network cost.

**Requirement#22:** It should be possible to access the contents of files efficiently according to the data model since datasets may be distributed among several database management systems.

### 3.4.3 Network-Related Requirements

In Grid environments the communication requirements of an application are subject to numerous network constraints. These considerations have a direct bearing on the performance of the application, and hence on how resources need to be scheduled for it. As in the HEP scenario described above, it is typical for an application to require access to significant amounts of input data from remote locations. Applications may also need to write data to a remote file. Distributed applications, including tightly coupled simulations, may need to regularly exchange data between distant machines. All of the above scenarios are basically network dependant operations and without an efficient and reliable network Grid applications may well under perform. Therefore, scheduling decisions should also consider the underlying network characteristics while placing jobs on the remote nodes (see requirement #19).

When selecting a dataset replica for Data Intensive scheduling, the underlying network between submission and execution sites as well as the link of the site storing the data plays an important role in scheduling optimization. Therefore the Scheduler should find a best network path between storage and computing elements (see requirement#14). Those sites should be given priority in replica selection which has better and stable networks (see requirement#4). The network is a key element in the Data Intensive scheduling and scheduling system should always calculate and incorporate the network measurements while planning job submission. This inclusion can lead us to Network Aware services and Network Aware scheduling decisions. Therefore the cost of computation, data transfer and network should be taken into account before making a job scheduling decision.

While communication-intensive applications may require high-speed networks for transferring large amounts of data, some applications simply require dependable networks. At

minimum, it must be possible to discover and monitor the conditions of networks on which applications are to be run. This prescribes the need for network monitoring and prediction services that can provide applications, resource management services, and users alike with the ability to query the characteristics of networks associated with the target computing resources (see requirement#16 and 18). For an end user, such characteristics may include bandwidth, RTT (latencies), packet loss and Jitter (reliability of the network). These parameters illustrate the state and quality of the network which are vital for the data intensive scheduling decisions. Such parameters may also provide details to applications about how to best use the network, for example the alternative route and location for sending or retrieving data in a smaller amount of time. Some applications require certain minimum guarantees on the network bandwidth or network speed of communication links between network nodes. They rely on some type of Quality-of-Service within the scheduling System providing guaranteed reservation of network capacity along with computational resources (see requirement#10).

**Requirement#23:** Network aware services are required for the data intensive scheduling decisions.

### **3.5 The Impact of the Use-Case Approach and Requirement Recommendations**

The use case approach proved an efficient and effective technique for collecting essential requirements from a group of users and helped to focus on their real needs. It helped to arrive at a common, shared vision about how the DIANA Grid Scheduler will really behave when it interacts with large amounts of data, computation and high performance networks. It also helped to avoid unnecessary and superfluous requirements. Some confusion may arise when people start thinking in terms of a real life use case. Proposed use case may actually be list of features, or desired behaviours or attributes [121]. However, we tried to map each of these requirements to the High Energy Physics use case and tried to avoid unnecessary features and behaviours. Use case methodology constituted a powerful, user-centric tool for the DIANA scheduling requirements specification process and are equally applicable whether the intended system is constructed using object-oriented techniques or in a more traditional environment.

Given below are the summarized requirements which will be followed and referred back to in the later sections of the thesis. All relevant requirements from sections 3.3 and 3.4 are grouped (G-1 to G-5) and will become the basis for future chapters. These requirements are aligned and complement the hypothesis and the research questions described in section 1.3 of chapter 1. Specific requirements recommendations which can be revealed from the requirement process for DIANA scheduling are given as follows:

- **G-1 Data Intensive Scheduling:** Data location should be given consideration for the scheduling decisions. Data should also be moved towards the jobs if required for scheduling efficiency. Data and jobs should be moved to a third location if feasible for scheduling and execution efficiency. See requirements # 1, 7, 12, 13.
- **G-2 Network Aware Scheduling:** The best network path from computing and storage elements should be decided and included in the scheduling decisions. The network is a key element and scheduling system should calculate and incorporate the network measurements in planning the job submission. Those sites should be given priority in replica selections which have better and stable networks. See requirements #2, 4, 14, 18, 23.
- **G-3 Scheduling Priority:** The priority of the job placement is vital while dealing with the very great frequency of data intensive jobs in bulk scheduling. There should be a mechanism to prioritize the jobs for providing quality of service to certain high priority jobs. See requirements# 5, 9, 10, 11.
- **G-4 Scheduling Cost:** Optimized scheduling decisions are not possible unless compute, data and network characteristics are brought under a single scheduling algorithm. See requirements# 6, 8.
- **G-5 Scheduling Hierarchies:** Decentralized and self organizing schedulers are required for bulk job scheduling. Scheduler should have the capability to steer and move the jobs to the sites having better resources. See requirements # 3, 16, 19.

These summarized requirements require detailed study, mathematical modelling, experimental and simulation tests and exploration. Problems like bulk job scheduling and data intensive scheduling in a Grid environment have not been adequately considered before. Storing, replicating and locating the petabytes of data and then scheduling millions of jobs to

access subset(s) of this data on thousands of compute resources worldwide is far from being a trivial challenge and is the subject of what will be explored concerning bulk scheduling in this thesis. Similarly, network aware services and network aware scheduling is a novel area and due to the nature of next generation applications, this is very innovative and timely as a domain of research. Moreover, self organizing and scalable schedulers are a state of the art research area and need due attention to enhance quality of service in application execution. These research requirements explain and elaborate the issues defined in the hypothesis (see section 1.3 chapter 1):

Data intensive bulk scheduling can be significantly improved by taking into consideration a combination of network, data and compute costs, as well as by implementing effective queue management and priority control.

Some of the requirements outlined in the requirement section are outside the scope of the research and as such constitute the future directions of this research work. Requirement# 15 is about parameterized schedulers which is again a major research issue in its entirety and therefore will not be covered in this thesis. Software heterogeneity and data model requirements#20, 21 and 22 are not the issues of immediate concern to DIANA scheduling and therefore will not be considered in this thesis. Requirement# 23 is more about inter-process communication which is not very relevant to DIANA scheduling.

### **3.6 Conclusions**

In Grid Computing, each component behaves as a service, performs autonomously and has a self-contained behaviour. These characteristics demand an approach which is iterative, adaptive and descriptive and this has led us to follow Use Case Modelling within the requirement engineering process. Waterfall models of system implementation have been demonstrated to be less successful in satisfying user requirements since it is not possible to address all the aspects of user involvement at the early stages of a research prototype. Use-case modelling technique has been adopted to identify and specify the user requirements for the Data Intensive and Network Aware Grid scheduling and to iteratively evolve the system. A set of requirements is captured and the system has been modelled using the use case methodology. Requirements have been extracted from a HEP use case as they appear and a set of requirements is recommended for the DIANA scheduling. It is revealed that data

location, network characteristics, overall cost of the job placement and a universal algorithm to manage computation, data and networks are the key requirements which should be incorporated in such a scheduling approach.

In the remainder of this thesis the requirements for a DIANA Scheduler are pursued. As a first step a set of algorithms is devised to address these requirements and their mathematical and theoretical description is given in chapter 4. These are crafted to optimize the scheduling process and take into account various greedy as well as economic approaches for the scheduling optimization. These algorithms in essence implement and incorporate the requirements which were short listed in the section 3.5.

## Chapter 4

### Scheduling Optimization Approaches

Chapter 3 provided an overview of the DIANA scheduling requirements and listed the salient features of such a system. Amongst those features it was established that data location is central to the working of an optimized Grid and hence it should be part of all scheduling decisions. It was also established that the ‘best’ or most optimal network path to computing and storage elements should be identified and that the scheduling system should calculate and incorporate network measurements while planning job submission. Chapter 3 explained that the network, computation and data transfer costs and the priority of job placement are vital when dealing with a large frequency of data intensive jobs and for optimizing scheduling decisions. In chapter 4 a description of the DIANA scheduling algorithm and its constituting key scheduling parameters is provided and how they influence the scheduling optimization is demonstrated. The scheduling problem is divided into different scheduling costs. These scheduling costs, namely the data transfer cost, the compute cost and the network cost, are derived through suitable mathematical formulae and are explained with the aid of an example. The queue management algorithm and related issues are described in section 4.5. After this, bulk scheduling is introduced and its association with priority and queues is established. The DIANA scheduling algorithm is extended to handle the Grid bulk job scheduling process.

#### 4.1 Introduction

There are three main phases of scheduling on a Grid:

- Phase one is that of resource discovery, which generates a list of potential resources.
- Phase two involves gathering information about those resources and choosing the best set to match the application requirements (the so-called “matchmaking” phase). This is where this thesis makes its main contribution.
- Phase three is the job execution phase including file staging and cleanup.

In this chapter, the research issues and approaches to optimize the matchmaking phases are described. In this second phase, the best match between jobs and resources is central. Many

heuristics have been proposed to obtain the optimal match as discussed in [124] but for the data intensive scheduling, the network characteristics need to be included in the scheduling decision as compared to the existing matchmaking process that does not take network parameters into consideration. Thus we need to embed the network information into the scheduling algorithm to improve the efficiency and the utilization of a Grid system. The overall goal is to minimize the computing time for applications which involve large-scale data.

There are two different goals for task scheduling: high performance computing and high throughput computing. The former aims at minimizing the execution time of each application and is generally used for parallel processing, whereas the latter aims at scheduling a set of independent tasks to increase the processing capacity of a system over a long period of time. We describe here a high throughput computing scheduling approach since most of the data intensive applications are very time consuming in execution and in data transfer operations.

## **4.2 Input Parameters and Objectives**

The following parameters will be used for the scheduling optimization related decisions. The reasons to include each of these parameters are discussed in section 4.3. The measurement and experimental process is explained in the later chapters of this thesis. These parameters have direct significance for Data Intensive and Network Aware Scheduling optimization and will be discussed later in this section:

- Bandwidths and latencies (RTT) of the network links.
- Packet loss and jitter
- Computing cycles available
- Site loads and respective job queues
- Job Priorities
- Size of the application executables
- Size of data files (input as well output)
- Location of the data files

A few characteristics are described which can help in creating an optimized scheduling algorithm. Utilization is the first performance criterion - we want to keep the CPU as busy as possible since, if the CPU is busy in exchanging processes, then work is being carried out. Another measure of work is the number of jobs completed per unit time and this is called the throughput. The interval from the time of submission to completion is termed the turnaround time and has a significant bearing on performance indicators. Turnaround time is the sum of the periods spent waiting to access memory, waiting in the ready queue, executing the CPU and doing input/output. The waiting time is the sum of the periods spent waiting in the ready queue. In an interactive system, turnaround time may not be the best criterion. Another measure is the time from the submission of a request until the first response has been provided. This measure, called the response time, is the time it takes to start responding but not the time that it takes to output that response. In the proposed scheduling algorithm, we want to maximize CPU utilization and throughput and minimize turnaround time, waiting and response time. A scheduling algorithm is created based on the measured parameters above which forms an important element of the matchmaking process in the Grid Scheduler. The following are the targeted metrics within the scheduling process by which the success and optimization level of the scheduling system needs to be gauged:

- Queue and waiting time
- Processing and execution time
- Input data transfer time
- Executable transfer time
- Results transfer time

The total time to execute a job in a Grid environment will be the sum of all of these times.

### **4.3 Cost Estimators**

There are three major cost estimates which need to be calculated for the scheduling algorithm: the network, computation and data transfer costs and scheduling optimization will be based on these three estimates. These estimates further depend on the parameters discussed in the following sections. These are explained in the following sections. The importance of each cost function can be adjusted by allocating weights to these parameters.

These weights are dynamically assigned and their allocation procedure is discussed in section 4.3.5.

#### **4.3.1 Network Cost**

By far the most important factor affecting scheduling process is that of network cost. The load, capacity and availability of network links used during data transfers may heavily affect the Grid application performance. Consequently, a fully functional Grid is critically dependent on the nature and quality of the underlying network, with connectivity, performance and security being key factors, as described by Primet et al. [125]. The high-bandwidth demand created by very large datasets necessitates the deployment of network infrastructures with efficient data transport capabilities. Consequently the new challenges posed by Grid applications lead to new research directions in network infrastructures, services, and data transport. Grid has given birth to distributed systems applications that are deployed on a wider scale (in some cases across continents). They need large data transfers for longer durations of time to fetch the data (even from one continent to another), computations run for long intervals of time to do the analysis on large datasets, resources are dynamic and may come and go at any time, (for example due to network or other failures), but at the same time the whole system works as a single entity to achieve some computation goal. This was not the case with existing distributed applications which were restricted in scale and requirements. These new applications, coupled with Grid toolkits, represent a paradigm shift in how applications interact with the network and with Grid middleware. Application usage of the network often requires a near-real-time (or even a real-time) information feedback loop on the available resources together with intelligent decisions on how best to take advantage of these resources. In order to provide the right quality of service (QoS) to Grid applications and hence scheduling, it is important to first understand how the network is performing and to determine the level of quality of service that currently exists in the network. This is measured using four variables, namely latency, dropped packets, throughput and jitter. The reasons to adopt these parameters are discussed in the following paragraphs of this section. These network parameters influence the data transfer and indicate the health of a network. For example, a network having less packet loss is better than the one having a higher packet loss. Since we are interested in the data transfer capability of the network, the term which is used to determine this capability is the TCP throughput. TCP

throughput can be obtained by combining the losses and the Round Trip Times (RTTs) using Mathis's formula [126] for deriving the maximum TCP throughput. Given the historical measurements of the packet loss and the RTT, we can calculate the maximum TCP bandwidth for a certain amount of time for various groups of sites. Mathis's formula describes a short and useful formula for the upper bound on the transfer rate:

$$\text{Rate} < \left( \frac{MSS}{RTT} \right) \times \left( \frac{1}{\sqrt{\text{loss}}} \right)$$

Equation 1: TCP throughput calculation

Where: Rate is the TCP transfer rate, MSS is the maximum segment size (fixed for each Internet path, typically 1460 bytes), RTT is the round trip time (as measured by TCP) and loss is the probability of packet loss. It is clear from the above equation that RTT, TCP throughput or bandwidth and packet loss (including out of order packets and duplicate packets) should be made part of the scheduling algorithm since it has to deal with large data transfers when scheduling data intensive jobs. One way of measuring the QoS is to measure the number of packets being dropped. Since IP is by nature unreliable due to its stateless nature, it can drop packets as the network becomes congested. A higher packet drop rate indicates a congested network, leading to a poor QoS. The packet loss is a good measure of the quality of the link for many TCP-based applications. Packet loss can occur for a variety of reasons including link failure, high levels of congestion leading to buffer overflow in routers, Random Early Detection (RED, a congestion avoidance algorithm which monitors the average queue size and drops packets based on statistical probabilities. RED makes Quality of Service differentiation impossible.), Ethernet problems like the frame errors, and the occasional misrouted packet. The main observable effect of packet loss is poor data throughput performance which is a prime concern in data intensive scheduling.

However, packet loss is not the only cause of poor performance, so care is needed in diagnosing whether genuine packet loss is being experienced. The response time or RTT is the second parameter that can give an idea of the 'ping' data rate (KB/s). Response time is the time it takes for data to travel from source to destination without any load. The RTT is related to the distance between the sites (since a long distance will lead to a greater number of hops along the path and this can increase the delays) plus the delay at each hop along the

path between the sites. RTT is usually used to measure latency. The RTT as measured says something about how quickly a small packet can be transferred from a backbone to a server site and back, but says nothing about how much information a server site can send in a given period. Moreover, for better QoS and network predictability, we also need to include the jitter in the scheduling algorithm. Jitter is an unpredictable variable delay in data reaching a destination from a source under certain loading conditions. It is unpredictable since it depends upon how congested the network is. Jitter becomes of critical importance in a converged network where voice, video, and data are being passed over the same link. Jitter is more important for streamed multimedia data compared to a Grid network which is being used for massive data transfer, for example, data transfer for physics analysis. However, jitter can be equally used to describe the reliability of a network being used for all types of applications since a higher jitter indicates congestion in the network or some other bottleneck or network anomaly. Therefore, as the network utilization increases, the number of dropped packets and the amount of jitter increase. Consequently, network cost is the combination of all of the above parameters. We assign weights to each value depending upon the importance of the parameters to calculate an aggregate value of the network cost:

$$NetworkCost \propto \frac{Losses}{Bandwidth}$$

where

$$Losses = RTT \times loss \times W_2 \times Jitter \times W$$

Equation 2: Calculation of the Network Cost

Where W is the weight assigned to each parameter depending upon the importance of the particular parameter. On the Internet, the network partitions a message into parts with a certain size in bytes. These are called the packets. A typical packet contains perhaps 1,000 or 1,500 bytes. Therefore loss is measured in bytes per second, RTT is measured in milliseconds and jitter is a number. Bandwidth is measured in bits per second and therefore network cost is in seconds. It can be in minutes and in extreme circumstances in hours if the network is performing poorly. All weights are assigned subject to a cost. A higher cost will lead to a higher weight and in some cases we can manipulate these weights to prioritize certain parameters in the algorithm. For example, if network cost is negligible, we assign bigger weights to this cost to influence the data intensive scheduling. But for compute intensive jobs

we do not change the values of the weights in the network cost rather we can manipulate the compute cost weights on the same rule as is described for data intensive jobs. A higher RTT indicates that a computation site is distant from the storage site where the data resides and therefore the cost to fetch the data would increase. We can increase the significance of this parameter, if required, by assigning a higher value to the associated weight.

Higher bandwidth reduces the cost of data transfer and hence the job execution. We can accommodate this behaviour by assigning a smaller value to its  $W_i$ . Furthermore, jitter is of less importance for data intensive applications with no significant cost involved as a result of higher or lower jitter. We can assign a minimal weight to this parameter but cannot ignore it completely since, if this parameter has a value higher than an acceptable figure, then this reflects a network bottleneck which is not ideal for data intensive transfers. In these circumstances, a higher value is required for this parameter to reflect this behaviour in the scheduling algorithm. The same is true for the packet loss: higher packet loss predicts a less reliable network, and we should give less importance to such a site when making scheduling decisions.

#### 4.3.2 Computation Cost

The second important cost which needs to be part of the scheduling algorithm is the computation cost. Jin et al. describe a mathematical formula [49] to compute the processing time of a job:

$$\text{Computation Cost} = \frac{Q_i}{P_i} \times W5 + \frac{Q}{P_i} \times W6 + \text{SiteLoad} \times W7$$

Equation 3: Formula for the Computation Cost

Where  $Q$  is the total number of waiting jobs on all the sites,  $Q_i$  is the length of the waiting queue on site  $i$ ,  $P_i$  is the computing capability of the site  $i$  (the total number of processors at site  $i$ ) and SiteLoad is the current load on that site. SiteLoad is calculated by dividing the number of jobs running by the processing power of that site. The  $Q_i/P_i$  ratio computes the processing time of the job. The  $Q_i/P_i$  ratio of the two sites cannot be the same since the number of jobs submitted to the sites will always be different due to differing SiteLoads and other appropriate parameters such as the data transfer cost of the sites. The unit of the computation cost is time (minutes or hours). Again  $W5$ ,  $W6$  and  $W7$  are the weights which

can be assigned depending upon the importance of the queuing and the processing capability. A higher weight means that a particular parameter is more significant for the scheduling decision. For example, a larger queue makes a site less attractive for job placement so we assign it a bigger weight to make the cost higher. Similarly, site load reflects the current load on a site, so again we assign a higher weight if the load on that site is higher. Higher computing capability is a desired feature for job scheduling on a site and this means the cost will be minimized if the computation is higher and we can assign a smaller weight to make this site favoured for job scheduling.

### 4.3.3 Data Transfer Cost

The third most important cost aspect in data intensive scheduling is the Data Transfer Cost (DTC) which includes input data, output data and executables. Park and Kim describe a mathematical technique [42] to calculate the aggregate data transfer time which includes all three parameters. Here we do not use bandwidth only to calculate the data transfer cost, rather we use the Network Cost (NC), as calculated in Section 4.3.1. We take the case of remote data and different (remote) execution sites so that the Meta-Scheduler can consider a worse-case scenario in scheduling. Network cost will be higher if the bandwidth of the WAN link between the two sites is small and vice versa. A higher network cost will lead to a longer data transfer time and therefore a higher data transfer cost. Similarly a larger data size will take more time to transfer and therefore the cost required to transfer this data will be higher. From this discussion, we can deduce that these two parameters (i.e. network cost and data transfer cost) are proportional to each other and therefore:

$$DTC \propto NC \dots\dots\dots (I)$$

Since data transfer cost will increase with the data size, we can also write equation I as

$$DTC \propto Data \times NC \dots\dots\dots (II)$$

From I and II, we can deduce that

$$DTC = W_8 \times Data \times NC \dots\dots\dots(III)$$

We can further expand the equation III into the following expression:

Data Transfer Cost (DTC) = Input Data Transfer Cost + Output Data transfer cost + Executables transfer cost

$$DTC = (InputData \times NC) + (OutputData \times NC) + (executable \times NC)$$

Equation 4: Data Transfer Cost

In this equation, we consider three elements for data transfer. All data transfer costs are basically the time consumed in transferring the data, therefore the units balance on both sides of the equation. Input data transfer cost is the most significant one since analysis is performed on the input data and it can be hundreds of terabytes. This huge data transfer and replication process strongly depends on the network cost. A higher network cost will increase the data transfer cost and we can use the associated weight to adjust this value according to its importance. For example, higher bandwidth will minimize the network cost which will reduce the input data transfer cost. The same is the case for the output data since output data needs to be transferred to the location from where the job is submitted. So output data transfer cost should also be calculated when making scheduling decisions and in some cases this data is even higher than the input data.

Executables mentioned in the data transfer cost are application data and user code which will be submitted for execution. This data is much smaller than the input and output data and therefore its associated transfer cost is lower compared to the input and output data transfer costs. In some cases it is necessary to calculate the additional cost due to the application data cost. Large executables have a significant transfer cost when remote execution sites are less reliable, for example due to poor WAN connectivity. This is also true for compute intensive jobs where computation nodes are not located at the same site. We can reduce the response time by moving input data from one site to an alternative that has a larger number of processors, since computational capabilities of a remote site without replicated data can be superior to the capabilities of other sites with replicated data. This is explained with the help of an example in section 4.4.2. In this scenario, the input data located in site *i* is transferred to site *j* which has sufficient computational capabilities. Also application codes should be transferred from the local site to site *j*. Then the processing is performed at site *j* and the resulting data will be transferred back to the local site *i*. If the data is available on tape, access latency can be higher and it can no longer remain feasible to move the data towards

the jobs. But this problem will remain equally valid if the job is scheduled to a local site since access latencies will remain the same. Therefore tape latencies should also get a weight to consider before the job is scheduled on a site. Some time it can be more feasible to fetch data from a remote site due to stable networks than to get the data from a tape which is available locally or on a close site.

#### 4.3.4 Total Cost

Once we have calculated the cost of each stake holder, the total cost is simply a combination of these individual costs as calculated in Sections 4.3.1, 4.3.2 and 4.3.3 thus:

$$\text{Total Cost } C = \text{Network Cost} + \text{Computation Cost} + \text{Data transfer Cost}$$

The main optimization problem that we want to solve is to calculate the cost of data transfers between sites (DTC), to minimize the network traffic cost between the sites (NTC) and also to minimize the computation cost of a job within a site. So for any particular job we calculate the data transfer costs across all sites and choose the one with the minimum network, data transfer and compute cost rather than minimizing these costs through changing the parameters. To simplify the optimisation problem we assume that any given site can have:

- One or many storage resources (Storage Elements, SEs)
- One or many computing resources (Computing Elements, CEs).

Therefore, we are mainly interested in the wide-area network performance rather than specifying all network details within a site. We assume that the local network latency is more or less homogeneous for all nodes (storage or computing) within a site.

#### 4.3.5 Allocation of Weights

Depending on the nature of the scheduling problem, it may be appropriate to give some of the cases greater weights than others in computing frequency distributions and statistics. The way we do this is to specify that a certain variable contains the *relative* weights for each case and should be considered as a weight variable. The goal of any weight is to prioritize or characterize different variables according to the chosen measure of contribution or influence. Since the objective in using weight allocations is to gauge relative weights rather than actual

weight values, arbitrary weighting schemes as discussed in the following paragraphs can attach incorrect weights to the component variable if care is not taken in assigning the weights. For example, network cost is given a higher weight for data intensive jobs and a lower weight in compute intensive jobs since compute intensive jobs do not require large datasets. However, if the network cost is not assigned a weight based on the job nature (compute or data intensive), it can lead to performance degradation for the Grid. Based on the assigned weights, the scheduler can assign a job to a site which may not be ideal for executing this kind of job.

Let us explain the philosophy of weight allocations through a worked example. Numbers are taken arbitrarily to explain the weight allocation in figure 4.1 but they are sufficiently representative of the actual Grid system and are closer to the real values. This will also demonstrate what are suitable weights for data intensive applications and how we can set the weight in case of high throughput applications. It is to be noted that the network compute, and data transfer costs mentioned in figures 4.1 and 4.2 are calculated using the equations discussed in sections 4.3.1, 4.3.2 and 4.3.3 respectively. Let us suppose we have a 100 GB of data located at a site in Japan and that we have jobs that need to access the data for processing and analysis. We also assume that we have only this single copy of the data across the Grid and that every job wherever submitted will use this data. Via an information service we have determined that a site in Switzerland has the greatest number of computing cycles available for the analysis since it is the least loaded available site and has fewest jobs in its queue. Moreover, there are 8 CPUs available in Japan and 50 in Switzerland, and the bandwidth between these sites is 100 MB/s. The queue size for the site in Japan is 20 whereas that in Switzerland is 2 jobs only. We also assume that the total jobs in the Grid are 1000 at this point in time.

We calculate the site load by dividing the jobs in the queue by the number of CPUs (assuming CPUs on all the sites have equal processing power). Now there are two options for data analysis: either we should submit the job to the site in Japan where the required data is available or we should transfer the data to the site in Switzerland where the computing capacity is available. The scheduler must decide where this job should be placed so that it has the least overall execution time. We check the compute cost and data transfer cost to enable the decision. The job is data intensive therefore data transfer should get a higher weight

relative to others. The scheduler assigns an equal weight to the compute cost on each site. The network cost should be ignored for the time being since we assume that it will remain constant between the two sites. Since we are making a pre-processing decision, we will take input transfer only. We can ignore the executable transfer cost since this data is minimal as compared to input data. RTT and Loss can be ignored since the network seems to be pretty stable. Normally a weight assigned to one variable will be the same for all sites otherwise we cannot compare the strengths or weaknesses of a site. Weights are arbitrarily assigned in the range 1 to 20 where 20 represents a very significant factor and 1 a very insignificant factor. In this example, the site load is a very important factor since it decides how long a job will wait until a job gets a CPU so we have allocated it a maximum value of 20. Further work is required to simulate the exact behaviour when we vary the weights for different variables and check the outcome on the scheduling optimization. Since, in this worked example, the data transfer cost for Japan is zero, we have assigned a minimal weight of 10 to the same variable for Switzerland site as otherwise it can bias the whole comparison.

Site	DTC	Compute Cost	Network	Total Cost
Japan	0	$10*20/8+5*1000/8+20*20/8$	0	700
Switzerland	$10*100*1024/100$	$10*2/50+1000*5/50+20*2/50$	20/100	10341.2

Figure 4.1: Weight Allocation and Cost

From the above calculation in Figure 4.1, it is clear that the cost for job placement in Switzerland is much higher than the one in Japan and we should send the job towards the data even if better computing cycles (CPU's) are available at the site in the Switzerland. We see that it is the data placement cost which has reduced the chances of selection for the site in Switzerland. Before actually placing the job on a site in Japan, we have to check if there is any other site where the cost combination is less. We have realized that although there is no better site than the one in Switzerland in terms of computation power, the Scheduler has found a site in UK where the bandwidth is much higher (10 Gbps) than the Japan-Switzerland link, and we should calculate the cost of job placement for this site. The site in

the UK has 30 CPU's available and there are 10 jobs in the queue. So we need to calculate the cost of job placement in the site in the UK.

We can see in this example that the better network link has enabled the scheduler to select a site other than those in Switzerland and Japan (cf. Figure 4.2) and the favoured solution is to move the data and the job towards this site in the UK. Although this site does not have the data and is not as powerful in compute resources as was the case in Switzerland, its job placement cost is still much lower than the other sites and clearly job placement on this site will reduce the overall execution time significantly.

Site	DTC	Compute Cost	Network	Total Cost
<b>Japan</b>	0	$10*20/8+5*1000/8+20*20/8$	0	700
<b>Switzerland</b>	$10*100*1024/100$	$10* 2/50 +1000*5/50$ $+20*2/50$	20/100	10341.2
<b>UK</b>	$10*100*1024/10*1024$	$10* 10/30 +1000*5/30$ $+20*20/30$	20/10*1024	<b>282</b>

Figure 4.2: Cost Calculation for best sites

#### 4.4 Scheduling Algorithm

In this section, the DIANA Scheduling algorithm is described which takes into account all the three costs which were discussed in the previous sections. The scheduling algorithm also takes into consideration the priorities but their role and functionality will be discussed in the sections 4.5 and 4.7. Following this a scheduling matrix is discussed to illustrate the DIANA scheduling process by incorporating the costs and making the site selection decisions. After this an example case study is described which further highlights the functionality of the Scheduling algorithm through a worked example.

##### 4.4.1 Pseudo Code of the Algorithm

There are two scheduling schemes that the proposed algorithm will use: a Normal Scheduling Scheme and Job Migration (see section 4.7). It is to be mentioned here that job migration and job export will be used interchangeably in this thesis. The Normal Scheduling Scheme deals

with those jobs which are being submitted to a site by the scheduler for the first time and have not, as yet, been migrated whereas migration based scheduling scheme deals with those jobs which are scheduled to a site as a result of some job migration process. Here, the meta-scheduler consults its peers, collects information about the peers (including network, computation and data transfer) and selects the site having minimum cost. It selects whichever site is the best site for its execution based on this cost estimation scheme. The meta-scheduler deals with both computational jobs and data intensive jobs using the DIANA meta-scheduling algorithm. As discussed in chapters 1 and 3, jobs under consideration are either simulation based or analysis jobs. Simulation jobs are compute intensive and analysis jobs are data intensive. In the job description file, the type of job is described which helps to differentiate between compute and data intensive jobs.

In the case of computational jobs (i.e. where the job requires mainly CPU time), the meta-scheduler should schedule a job to the site where the computational cost is a minimum. At the same time, we have to transfer the job's files so that the job can be transferred as quickly as possible. The job might also require some input data which suggests selecting a site which has better network capacity (i.e. highest response time and lowest latency). Therefore, the meta-scheduler will select the site with minimum computational cost but also takes into account the data transfer cost.

In the case of data intensive jobs, our preferences will change. In this case our job has more data and fewer computation requirements, and we need to identify the site where data can be transferred quickly and where computational cost is also low. In this case, data location will play an important role since the data transfer cost will be the key element in such a scheduling decision. In most cases jobs are at the same time both compute as well as data intensive and will most likely follow the third category of the algorithm. In the third category the algorithm considers compute cost, network cost, data location and data transfer costs, and the site having minimum aggregate cost is selected for job execution.



Figure 4.2-a: State diagram of the DIANA scheduling approach

The state diagram of the DIANA scheduling approach is shown in the figure 4.2-a and the algorithm works in the following manner:

```

If the job is compute intensive then
computationCost[] = getAllSitesComputationCost();
arrangeSites[] = SortSites(computationCost); //it will sort array in ascending order
for i=1 to arrangeSite.length
    site = arrangeSite[i]
  
```

```

        if ( site is Alive) send the job to this site
    end loop
end if
Else if the job is data intensive then
    dataTransferCost[] = getAllSitesDataTransferCost();
    arrangeSites[] = SortSites ( dataTransferCost ); //it will sort array in ascending order
    for i=1 to arrangeSite.length
        site = arrangeSite[i]
        if ( site is Alive) send the job to this site
    end loop
end else-if
Else if (job is data intensive and compute intensive)
    computationCost[] = getAllSitesComputationCost()
    dataTransferCost[] = getAllSitesDataTransferCost()
    NetworkCost[] = getAllSitesNetworkCost()
    // since length of computationCost and dataTransferCost array is same. So we can use any
    of them
    siteTotalCost [] = new Array[computationCost.length]
    for i = 1 to computationCost.length
        siteTotalCost [i] = computationCost[i] + dataTransferCost[i] + NetworkCost[i]
    end loop
    sites [] = SortSites(siteTotalCost)//ascending order
    for j = 1 to sites.length
        site = sites[i]
        if ( site is alive) schedule the job to this site
    end loop
end else-if

```

#### 4.4.2 Scheduling Matrix

We can now calculate the cost of the job placement on each site with respect to the submission site. This will be a relative cost since it will always be measured with reference to the user's location on the Grid which is in fact the submission site. This scheduling matrix is constantly updated at each site and passed between the site meta-schedulers to determine the Grid weather at each site and to exchange the load and other related information between the sites. Next, we can populate a cost matrix with cost values against each site. In detail we look at the number of possible sites and calculate the total cost for each pair (site i – site j) and put that into our cost matrix.

If there is a local scheduler at a given site, the cost matrix will be a 1xN matrix, whereas in the case of a Meta-Scheduler, it will be an NxN matrix. Since our intended target is the global or Meta-Scheduler, we shall consider the latter case. We do not take into consideration

all the computation and storage sites in this cost matrix since that would require significant effort in calculating the cost of each site against all others in the Grid and the matrix optimization itself requires further research. Instead, we rank the sites on the basis of storage and computations cost and select the best sites (five in this example), which are then used to populate the matrix. It is ensured that these “best” sites have the least cost of all sites in the Grid since this can promise a reduced time for the overall job execution and ultimately will lead towards an optimized scheduling and an optimized Grid.

Figure 4.3 shows the cost matrix giving the overall cost of job submission from one site to all others in the Grid.  $C_{ij}$  is the cost of a particular site  $i$  from any other one  $j$  in the Grid and the matrix therefore shows the total cost as calculated above for each site in the Grid. We assume that a user can submit a job from one location to any other one in the Grid. Each site has almost zero data transfer and network cost with reference to itself and this is shown by blank blocks in the matrix. This is the case when data is available locally; it will of course still have a computation cost. If this computation cost is high as compared to the total cost at any other site, then the job will be submitted to the one having the lowest cost. Whether a single job is being submitted to the scheduler or bulk job submission is being managed by the Meta-Scheduler (as will be discussed in the later sections of this chapter) the cost matrix is equally valid since the cost mechanism will describe the time and cost of each job in the global perspective.

	Site 1	Site 2	Site 3	Site 4	Site 5
Site 1		$C_{xy}$	$C_{xz}$	$C_{xw}$	$C_{xf}$
Site 2	$C_{yx}$		$C_{yz}$	$C_{yw}$	$C_{yf}$
Site 3	$C_{zx}$	$C_{zy}$		$C_{zy}$	$C_{zf}$
Site 4	$C_{wx}$	$C_{wy}$	$C_{wz}$		$C_{wf}$
Site 5	$C_{fx}$	$C_{fy}$	$C_{fz}$	$C_{fw}$	

Figure 4.3: Cost Matrix for the Meta-Scheduler

There are certain cases where jobs are submitted in bulk, but bulk job submission does not imply that all the jobs will be scheduled on a single site since each job group will have different computation and data requirements and hence their associated cost will also be different. If the bulk jobs are not resource intensive, it may not be a problem to schedule all the bulk jobs on a single site, otherwise it may be chaotic to submit such jobs on a single site as this will generate long queues and could result in an overloaded site and a sub-optimal scheduling strategy. A long queue will not be a problem if there is a large CE with many CPUs where long queues can be served quickly. A long queue and a fast machine is much better than a short queue on very slow machines since all the jobs in the queue will be quickly consumed by the faster CPU's. The cost model takes this aspect into account when we talk about computing capability and this can be adjusted by assigning suitable weights to each cost factor. In the case of bulk jobs, where many jobs are simultaneously submitted for scheduling, if the bulk submission shares common input data, then these can be scheduled on a single site. If the cost of the data transfer is more than the cost of the computation, the bulk submission of the jobs will be cost effective for scheduling on one particular site. A detailed algorithm on bulk scheduling is discussed in section 4.6.

The cost matrix is equally valid for other types of jobs as well. For example, computation intensive jobs can also profit from this matrix and we can populate the matrix with the computation cost of that particular site as detailed in Section 4.3. In this case, although the data transfer and network cost is minimal, queue and computation costs can be helpful for making optimal scheduling decisions where a suitable combination of queue length, site load and computation cost can yield higher throughput. However, the most important use of the scheduling algorithm based on this cost matrix is for data intensive jobs. From the preceding discussion it is evident that this cost matrix can be helpful in finding the cost of job submission on a particular node from one point to any other point on the Grid. The cost calculation is relative since each cost element is between any two sites, which is the function of the different parameters including network, data and computation cycles available.

Once the cost matrix is populated, we can find the minimum cost of a particular site from all others sites by searching the cost matrix. This will indicate the site having the least cost. Thus:

$i$  = Site Number

```

C[i][j] represents an nxn matrix containing costs of site i with site j
minimumWeight = C[i][1]
siteIndex = 1
for j := 2 to n # n represents total number of sites
if( C[i][j] < minimumWeight)
siteIndex = j
minimumWeight = C[i][j]

```

Once the site with the least cost has been selected, the resource broker (i.e. the submission and execution service) will submit the job on that site. The cost matrix is the core of the DIANA Scheduler in selecting the optimal site for job execution. This cost matrix integrates all the costs which are calculated using the parameters such as queue and waiting time and the data transfer time etc. as discussed in section 4.2 and is used by the sites to exchange information (compute, network and data transfer costs etc) between the sites. Each site can be evaluated for the job scheduling using its cost matrix values with respect to other sites. The network cost calculated in the algorithm is used to select the best replica of a dataset and this cost will be used as input to the scheduler. Once the broker gets the best replica and its location, it is asked to bind one of them with a suitable computing element for the specified job. For this, computation and data transfer costs are used to find a best combination of computing and storage element. Finally, the submission service submits the job to the particular site which is selected by this algorithm. This is explained with the help of an example in the following section.

#### 4.4.3 An Example Scheduling Matrix

In the example diagram shown in the Figure 4.4, we have selected the five best sites in the example Grid environment in terms of their costs which are located in Italy, Austria, Switzerland, UK and Japan. We further suppose that these sites are ranked best on the basis of their network, storage and computation costs. Now we populate the matrix by taking the cost of each site from the rest of the sites. We also assume that local sites are not always the best for job execution since in some cases the data is not available locally (therefore, we do not include these values in the cost matrix). The rows are the sites where the jobs are executed and the columns are the sites from where we have to submit the jobs. Now, we have populated the cost matrix for these best sites in the Grid and filled the blocks with the test

values of the costs from each node to all others. Ideally, network and data transfer costs remain the same from one node to another measured from any of the two locations and we have used this assumption while populating the matrix. However, computation cost will not be the same since each site has a different computing capability and as a result, the cost from Italy to UK and vice-versa will not necessarily be the same. Once the matrix is populated, we have a fair idea of the cost of job submission and execution. Now we can easily decide which site is the best for job submission. From the matrix, it is evident that if a job is submitted from the site in Switzerland to the site in the UK, it will be the most efficient case of job execution (Figure 4.5). Similarly, we can decide on which site will be optimal to execute a job from any other one in the matrix. The weights assigned are dynamic and can change with time. For a static set of the results, these weights will remain constant. Since monitoring tools can measure the cost parameters quite frequently, we will also update these weights accordingly.

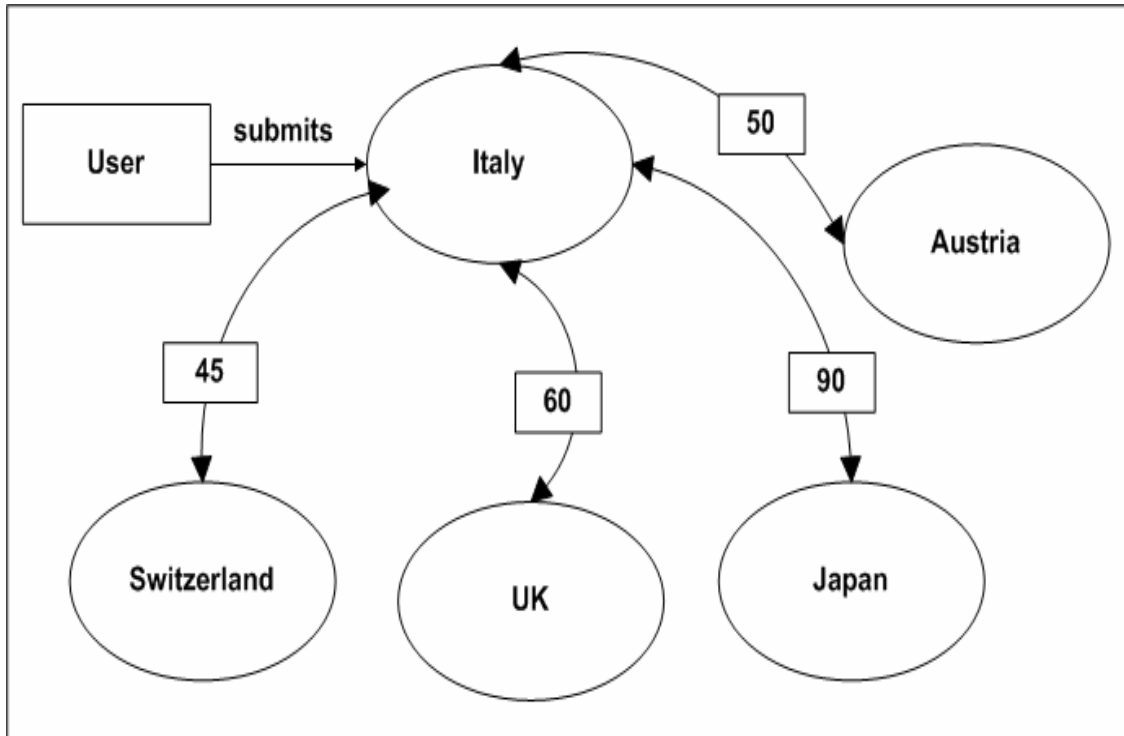


Figure-4.4: An example scheduling scenario

	Italy	Austria	Switzerland	UK	Japan
Italy		50	45	60	90
Austria	58		48	65	72
Switzerland	64	42		38	85
UK	72	65	50		65
Japan	70	72	85	65	

Figure 4.5: Example Scheduling Matrix

## 4.5 Queue Management in DIANA

### 4.5.1 Multilevel Queue Scheduling

In Grid, users submit a number of jobs which in most cases cannot be executed in real time. They are most likely to have to wait before an execution slot becomes available. Scheduling queues are the mechanism where these jobs wait and in most cases, such as the LCG and OSG project, job queue times can be larger than the execution times. It is important to have a mechanism for managing the scheduler queues so that users not only get an acceptable level of quality of service but overall wait times are also reduced. Moreover, there should be a mechanism so that all users should get an execution slot for their jobs instead of allowing the jobs to spend considerable time (days in some cases) in queues. This should also discourage the monopolistic use of the resources by restricting the users to submit bursts of jobs. To address these concerns, the meta-scheduler supports multilevel queue scheduling and its scheduling algorithm schedules the jobs following the queue management policies. For this, a queue management module is created which will be discussed in the following sections of this chapter. Moreover, as a result of the different quality of service requirements from users, jobs can be classified into different queues. For example, a common division is made between interactive jobs and batch jobs. These two types of jobs have different response-time requirements, and so might have different scheduling needs. In addition, interactive jobs may

have priority over batch jobs. A multilevel queue-scheduling algorithm partitions the ready queue into multiple separate queues as shown in Figure 4.6.

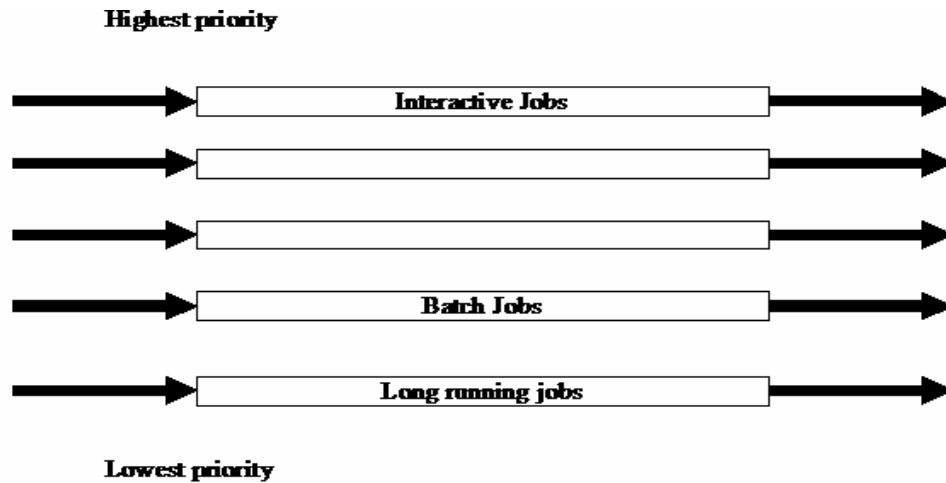


Figure 4.6: Multi level queue scheduling

The jobs are assigned to a queue, based on some property of the process, such as memory size, process priority or process type. Each queue has absolute priority over lower-priority queues. No job in the long running jobs queue for example, can run unless the queues for interactive jobs and batch jobs are all empty. In general, a multilevel feedback queue scheduler is defined by:

- The number of queues
- The nature of the jobs
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher-priority queue
- The method used to determine when to demote a process to a lower-priority queue
- The method used to determine which queue a process will enter when that process needs service.

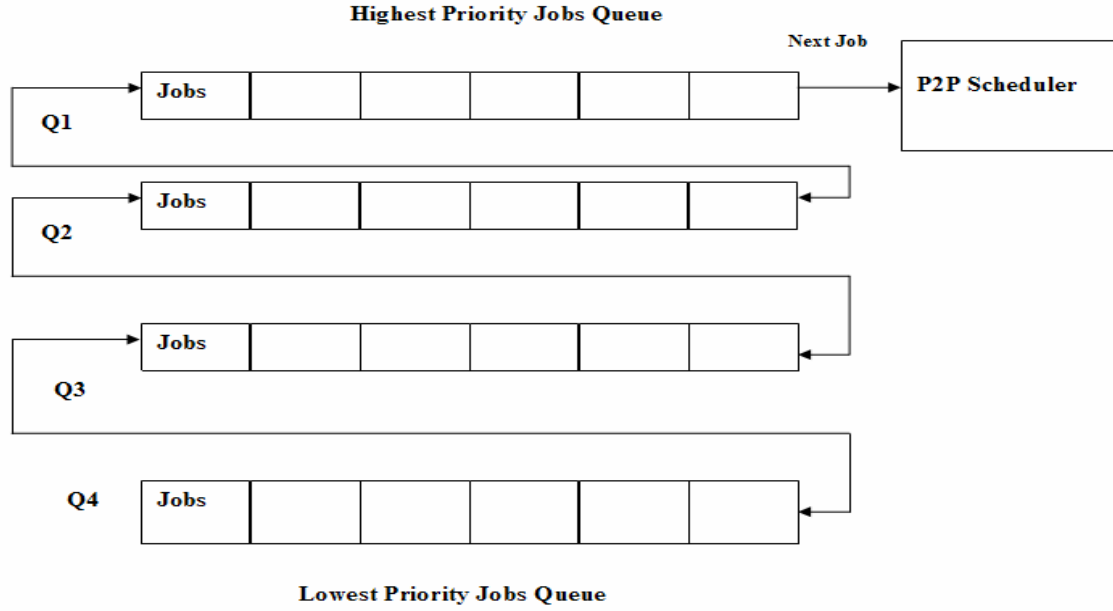


Figure 4.7: Multi-level feedback queues

In a multilevel queue-scheduling algorithm, jobs are permanently assigned to a queue on entry to the system. The Scheduler assigns priorities to jobs upon submission and the jobs join one of the queues based on the priority. The priority is assigned using a queue management algorithm as discussed in section 4.5.2. We have employed multilevel feedback queue scheduling as shown in Figure 4.7 since it allows a job to move between queues. The idea is to separate processes with different requirements and priorities. If a job uses less CPU time, as determined by the CPU requirements of jobs specified in the job description file or is very data intensive, it will be moved to a higher-priority queue. Similarly, a job that waits too long in a lower-priority queue may be moved to a higher-priority queue.

#### 4.5.2 Queue Management Algorithm

As discussed in the previous section, queue management is required to provide better management of the user jobs. Elmroth and Peterg [48] describe a Grid wide fair share scheduling system for local and global policies. They feature quota based scheduling and multilevel queues, although they do not consider reprioritisation and further, their solution was not P2P oriented. The GridWay Scheduler [70] provides dynamic scheduling and opportunistic migration but its information collection and propagation mechanism is not

robust and in addition it has not as yet been exposed to bulk scheduling of jobs. In DIANA we propose a multi-queue, feedback-oriented queue management approach for bulk job scheduling. Users may send jobs in a burst, and the meta-scheduler has to place all these jobs in queues after assigning priorities. We must ensure that the priority of the jobs *decreases* as the number of jobs in the queues from a particular user *increases*. This is important otherwise a single user may send thousands of jobs in a burst and thereby improve the priorities for all his jobs. Each queue will contain jobs having priorities falling in its specified priority range. According to our priority calculation algorithm, the priority of all the jobs will be in the interval  $\{-1, 1\}$  where -1 indicates the lowest priority and 1 indicates the highest priority. This range comes from the mathematical derivations in the following paragraphs and according to these equations, the priority will always be in the interval  $\{-1, 1\}$ . Therefore, the priority ranges for the all the queues, (suppose four queues Q1, Q2, Q3, and Q4) is proposed to be:

$$\begin{aligned} Q1 : 0.5 &\leq \text{priority} < 1 \\ Q2 : 0 &\leq \text{priority} < 0.5 \\ Q3 : -0.5 &\leq \text{priority} < 0 \\ Q4 : -1 &\leq \text{priority} < -0.5 \end{aligned}$$

In the process of selecting the job's position in the queue, we place the jobs in the descending order of their priorities i.e. the job with the highest priority will be placed first in the queue and a priority order is followed for the remainder of jobs. Finally we determine all those jobs having the same priority, and arrange them on a FCFS basis.

Job migration between queues is an essential feature of DIANA Queue Management. Jobs are dynamically assigned to the queues based on the assigned priorities. Further there is migration of jobs from lower priority to higher priority queues (and vice versa) and between the meta-scheduler queues at various sites which is an unique feature of the DIANA scheduler. Once jobs have been assigned by the meta-scheduler to a local scheduler, these jobs then cannot be migrated since these are beyond the control of a meta-scheduler. Moreover, a time threshold and a job threshold which dictate the priority control of jobs from a user lead to a suitable quality of service since each user can get a time slot and no user can be monopolistic in executing the jobs. On the arrival of each new job, all the jobs already present in the queues are re-prioritized. The re-prioritization algorithm may result in the migration of jobs from low priority to high priority queues or from high priority to low

priority queues. The reprioritization technique militates against aging since the jobs are assigned new priorities on the arrival of each new job and each job gets its appropriate place in the queues according to the new circumstances. In the case of congestion in the queues, the Queue Management algorithm will migrate the jobs to any other remote site where there are fewer jobs waiting in the queues. However, only low priority jobs are migrated to remote sites because low priority jobs (e.g. for a job falling in Q4) will have to wait for a long time in the case of congestion. Knowing the arrival rate and the service rate of the jobs, we can decide whether to migrate the job to some other site or not. The formula to decide whether there is congestion in the queues or not is:

$$(Arrival\ Rate - Service\ Rate) / Arrival\ Rate > Thr_s$$

Where  $Thr_s$  is the threshold value configurable by the administrator. If we increase  $Thr_s$ , then this means that the arrival rate may exceed the service rate and we must allow more jobs in the queues and there is less migration. In any case this value lies between  $\{0, 1\}$  interval. Taking this, we can now explain the queue management algorithm.

Suppose ‘ $n$ ’ is the total number of jobs of the user in all job queues. Let the new job require ‘ $t$ ’ processors for computation and ‘ $T$ ’ be the total number of processors required by all the jobs present in all job queues. We denote the quota of the user, submitting the new job, by ‘ $q$ ’ and the sum of the quotas of all the users, currently having their jobs in the job queues including ‘ $q$ ’, by ‘ $Q$ ’. So if the new user has already some jobs in the job queues, ‘ $q$ ’ will appear just once in the ‘ $Q$ ’. Let ‘ $L$ ’ be the sum of lengths of all job queues i.e. the total number of jobs present in all job queues including the new job. So if there are already, say, 1500 jobs in the job queues when 100 new jobs arrive, then  $L = 1600$ . To assign a new job a place in the job queue, we associate a number to it. This number is called the “Priority” of the job and has its value in the interval  $\{-1, 1\}$ . The rule is that “the larger the priority, the better the place will be”. Obviously if its priority is in the range  $\{0,1\}$ , it will be considered as a good candidate. To attain a good priority we must meet the following two constraints:

$$\frac{n}{L} \leq \frac{q}{Q} \text{ and } \frac{1}{L} \geq \frac{t}{T} \dots\dots (IV)$$

$$\text{Or } n \leq \frac{(q \times L)}{Q} \text{ and } L \leq \frac{T}{t} \dots\dots (V)$$

Combining these two inequalities IV and V, we get

$$n \leq \frac{(q \times T)}{(Q \times t)} \dots \dots \dots (VI)$$

We denote  $\frac{(q \times T)}{(Q \times t)}$  by 'N'

Equation 5: Queue Management Equation

'N' represents the threshold and obviously, it is dynamic. For each job, its value will be different. If a user's number of jobs in the queue crosses this threshold then the priority of the jobs crossing the threshold 'N' must be lowered .To calculate the priority of the new job, we use the following algorithm:

$$\begin{aligned} & \text{If } (n \leq N) \\ & \quad Pr(n) = (N - n) / N \\ & \text{Else} \\ & \quad Pr(n) = (N - n) / n \end{aligned}$$

Where Pr (n) denotes the priority of the new job. Also note that the priority will always lie in the interval {-1, 1}.

User	Quota (q)	Job's Processors Requirement (t)	Total # of Processors Required by all queues (T)	User's Total # of Jobs (n)	Total # of Jobs in all queues (L)	Quota Sum off all distinct users (Q)	$N = (q * T) / (Q * t)$	Priority Pr(n)
A	1900	1	1	1	1	1900	1	0.0

Figure 4.8: Priority Scenarios for a Single Job

On the arrival of each job, the priorities of all the other jobs will be recalculated. This technique is known as reprioritization. The reason for doing this is that we want to make sure that the jobs encounter minimum average wait time and the most “deserving” job in terms of quota and time is given the highest priority. Moreover, by using this strategy we need not

worry about the starvation problem and there is no aging since jobs are reprioritized on the arrival of each new job. The algorithm to reprioritize the jobs is the same as that mentioned above. The value of  $q$  for a particular user's jobs remains the same,  $Q$  and  $T$  remain the same for all the jobs, however,  $t$  is job specific and it may vary with each job. Therefore, the value of ' $N$ ' differs for each job. By using the above mentioned formula, we can calculate the priority for all the jobs and place them in their respective queues. Of course, if more than one job shares the same priority then the timestamp associated with each job is compared and the older job, which has spent more time in the queue, is placed before the new job. Also note that when a job is taken out of service the rest of the jobs need not be reprioritized.

Let us consider a scenario through the following worked example where a new job is submitted by user A and it requires one processor i.e.,  $t = 1$ . Experimentation details of this will be given in chapter 8 through actual deployment and simulations. We assume that the quota  $q$  for user A is 1900 and currently there is no job in the queue therefore,  $L = 1$ ,  $n = 1$ ,  $Q = 1900$  and  $T = 1$  and  $N = (1900 * 2) / (1900 * 2)$ . If we put these values in the algorithm and the test 'if' condition is true, then this job is placed in  $Q_2$ . This scenario is shown in Figure 4.8. We assume that the first job has not as yet been serviced and meanwhile, user A submits his second job demanding 5 processors i.e.  $t = 5$ , then  $L = 2$ ,  $n = 2$ ,  $T = 1 + 5 = 6$ ,  $q = 1900$ ,  $Q = 1900$  and  $N = (1900 * 5) / (1900 * 3)$ .

User	Quota (q)	Job's Processors Requirement (t)	Total # of Processors Required by all queues (T)	User's Total # of Jobs (n)	Total # of Jobs in all queues (L)	Quota Sum off all distinct users (Q)	Priority Pr(n)
A	1900	1	6	2	2	1900	0.6666
A	1700	5	6	2	2	1900	-0.4

Figure 4.9: Priority Scenario 2

Again putting these values in the algorithm, we find that the 'if' condition becomes false and  $Pr(n) = -0.4$  and therefore the job is placed in  $Q_3$ . Reprioritization then starts and the priority of the job(s) already present in the queue is/are recalculated. This time the priority is set to 0.666666 and this job is migrated from  $Q_2$  to  $Q_1$  i.e., the highest priority queue as shown in

the Figure 4.9. This is of interest because user A has submitted only two jobs and the threshold has not been exceeded on the second job. The algorithm equally handles all the users and jobs and the priorities decrease as the number of jobs by a user increases and it does not matter that the second job exceeds the threshold. Now suppose that another user B submits his first job which requires one processor i.e.  $t=1$  having user quota of 1700,  $q=1700$ . Assuming that the two jobs by user A are still in the queues,  $L=3$ ,  $n=1$ ,  $T=1+5+1=7$ , and  $Q=1900+1700=3600$ . The ‘if’ condition holds true and  $Pr(n)=0.6974$  and therefore the job is placed in  $Q_1$ . Reprioritization starts and as the result, the priorities of the previous jobs change and the first job by user A is migrated from  $Q_1$  to  $Q_2$  and the second job by user A is migrated from  $Q_3$  to  $Q_4$ . This is illustrated in Figure 4.10.

User	Quota (q)	Job's Processing Requirements (t)	Total# of Processors required by all queues (T)	User's total # of jobs (n)	Total # of jobs in all queues (L)	Quota Sum of all distinct users (Q)	Priority Pr(n)
A	1900	1	7	2	3	3600	0.4586
A	1900	5	7	2	3	3600	-0.6305
B	1700	1	7	1	3	3600	0.6974

Figure 4.10: Priority calculation for jobs from different users

It is notable that the first job by both user A and B demands 1 processor and the quota of user A is greater than user B, even if the priority of user B's job is greater than the user A job. This is because user A has submitted more jobs than user B and the algorithm handles this while calculating priorities. In this way the algorithm manages and updates the queues on the arrival of each new job.

#### 4.6 Bulk Scheduling

Scientific analysis tasks can involve thousands of compute, data, and network resources. Individual user analysis tasks, so-called bulk jobs, are generally characterized by the jobs consuming large amounts of these resources running for relatively short times, i.e. minutes to

a few hours. These jobs may involve an iterative process with the result of one analysis pass being used to adjust conditions for the next. For this reason turnaround is in general important. Such jobs can be either I/O intensive or can be CPU-intensive. In either case, such jobs tend to subject the computing system to very ‘spiky’ loads in CPU utilization and/or I/O. Since the datasets used in bulk jobs are of modest scale and are generally accessed multiple times, moving the data and caching it at the wide-area location of the available CEs is a useful strategy. In the following sections, a detailed analysis of the bulk scheduling process is provided and various approaches are discussed to optimize the bulk scheduling.

#### **4.6.1 Bulk Scheduling with DIANA**

Given the large number of jobs that can result from the job-splitting in the bulk scheduling, it should be possible to submit the job clusters to the scheduler as a unique entity, with subsequent optimization in the handling of the input data. Jobs may compete for scarce resources and this can distribute the load disproportionately among the Grid nodes. The scheduling approaches which were discussed in the previous sections of this chapter are based on ‘greedy’ algorithms in which a job is submitted to a best resource without assessing the global cost of this action. However, this may lead to a skewed distribution of resources resulting in large queues and performance and/or throughput degradation for the remainder of the jobs. We present a scheduling approach which not only allocates best available resources to a job but also checks the global state of jobs and resources so that the strategic output of the Grid is maximized and no single job can undergo starvation. In the following sections of this chapter, the DIANA Scheduling approach is extended for scheduling optimization of bulk jobs and an algorithm is explained for scheduling the bulk jobs. It is shown that a priority driven multi-queue feedback based approach is the most feasible to tackle the issue of bulk scheduling.

#### **4.6.2 Priority and Bulk Scheduling**

The proposed scheduling algorithm described later is termed a priority algorithm. A priority is associated with each process and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled on a First Come First served (FCFS) basis. Scheduling can be discussed in terms of high priority and low priority. Priorities can be defined either internally or externally. Internally defined priorities use some measurable

quantity or quantities to compute the priority of a process. For example, time limits, memory requirements, the number of open files and the ratio of I/O to CPU time can be used in computing priorities. External priorities are set by criteria that are external to the scheduling system such as the importance of the process. Priority scheduling can be either pre-emptive or non pre-emptive. The bulk scheduling algorithm described here is not a pre-emptive one. It simply places the new job at the head of the ready queue and does not abort the running job.

#### **4.6.3 Bulk Scheduling Algorithm Characteristics**

By following all of the above points, a multilevel feedback queue and priority-driven scheduling algorithm for bulk scheduling is proposed in the following sections. The salient features of this bulk scheduling algorithm are now briefly discussed. The pseudo code of this algorithm is given in Appendix-1.

- Job priority is important while scheduling the bulk jobs. High priority jobs are executed first. The priority of jobs starts decreasing if the number of jobs from a user/site increases beyond a certain point. It becomes less than all the jobs in the queue if job frequency (no of jobs per unit time) is very high.
- A priority scheduling algorithm may leave some low priority processes waiting indefinitely for the CPU (so-called “starvation”). We use an aging technique to overcome this problem. Starvation of the resources is controlled by controlling the priority of the jobs. This is shown in Figure 4.11. If no other job is available in the queue then all jobs from the user/site will be executed as high priority jobs. We give less preference to quota and accounting since this restricts the users to a particular limit. Instead we prefer priority to schedule bulk jobs and to control the jobs frequency as well as the queue. Similarly we do not follow the budget and deadline method of economy-based scheduling since the Grid is dynamic and volatile and a deadline is feasible only for static types of environment.
- It is assumed that all of the bulk jobs in a single burst will be submitted at a single site. The reasons behind this approach are obvious. Bulk jobs have almost similar execution and input requirements and work on the same set of datasets. Scheduling them on a single site will reduce the data transfer time and consequently will improve the overall execution time. If data and computing capacity is available at more than

one site job splitting and partitioning can be used. Queue length, data location, load and network characteristics are key parameters for making scheduling decisions for a site. The priority of the burst or bulk jobs from a single user is always the same since each such collection of jobs has the same execution requirements.

- We have implemented non pre-emptive scheduling [132] at the user level, since at the user level we cannot interrupt a running process and block it in order to allow a new process to run. Upon receiving a request, the policy decides whether to process the request immediately, or to postpone the execution. Another reason for not using a pre-emptive scheduling approach is the interactive nature of most of the jobs. Since most jobs are data intensive, this makes it increasingly important to consider the non pre-emptive mode as a primary approach due to the nature of long running data transfer operations as low profile processes. A ‘Round Robin’ approach inside queues is not feasible in this case since most of the analysis jobs are interactive and the user is eagerly awaiting the output. Any delay in the output may lead to an unsatisfied user and this requires us to provide additional resources to this job for quick processing. This also leads to the conclusion that the pre-emptive approach is not feasible for interactive jobs but can be considered for batch jobs. In this algorithm we consider only the interactive jobs used for a Grid-enabled analysis.

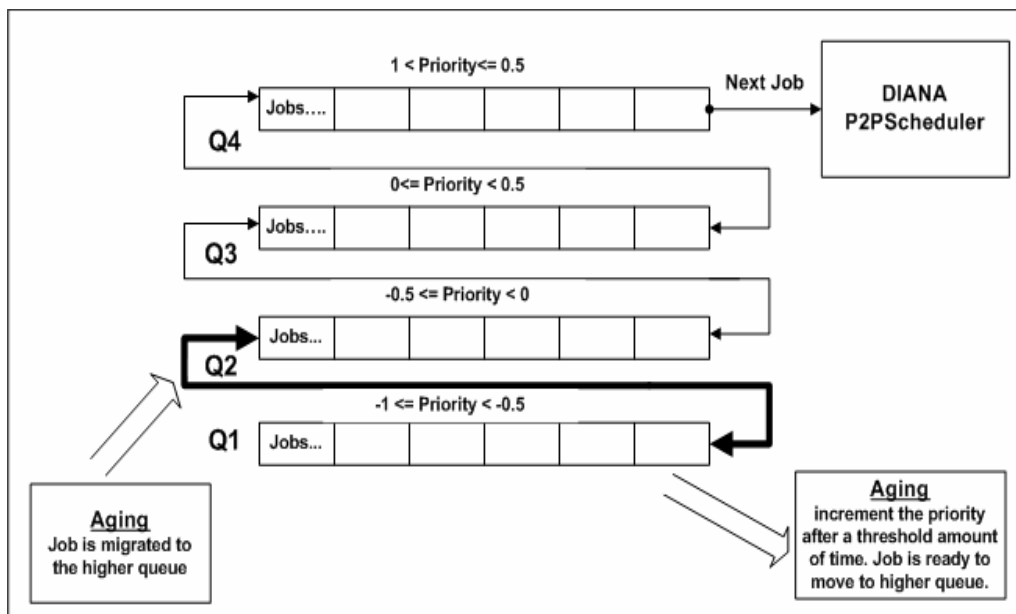


Figure-4.11: Multi-level Priority Queues and Aging on each site

- Job movement between priority queues is a key point of the algorithm. Jobs can move between low priority to high priority queues depending upon the number of jobs from each user and the time passed in a particular low priority queue. Although migration of the jobs between queues is supported, within a single queue we use the FCFS algorithm. Before jobs are placed inside the queue for execution, the algorithm arranges the jobs using the Shortest Job First (SJF) algorithm.
- The numbers of jobs in a bulk job submission are specified in the job description language. Fewer processors required means the job execution time is shorter and its priority should be set higher. All shorter jobs are executed before longer jobs; this reduces the average execution time of all jobs.
- Priorities can be of three types: user, quota and system centric. We employ a system centric policy (embedded inside the scheduler) since otherwise users can manipulate the scheduling process. In this way a uniform approach will be set by the scheduler for all users and a similar priority will be applied to all stake holders. Furthermore, priorities can be static or dynamic. Static priorities are assigned at the time of creation, while dynamic priorities are based on the processes' behaviour while in the system. For example, the scheduler may favour I/O-intensive tasks so that expensive requests can be issued as early as possible. If data fetching is a long operation then the high priority job can be put on hold until the required data is available to that site. During this data taking time interval, other jobs get the choice of execution. As soon as the required data becomes available, the job on hold is the first one to get the execution chance.
- Knowing the job arrival rates and execution capacity, we can compute utilization, average queue length, average wait time and so on. As an example, let  $N$  be the average queue length (excluding the jobs being serviced), let  $W$  be the average waiting time in the queue, and let  $R$  be the average arrival rate for new jobs in the queue. Then, we expect that during the time  $W$  that a job waits,  $R*W$  new jobs will arrive in the queue. If the system is in a steady state, then the number of jobs leaving the queue must be equal to the number of jobs that arrive.

$$N = R * W$$

Equation 6: Little's Formula

- This equation, known as Little's Formula [129], is valid for any scheduling algorithm and arrival distribution. We can use it to compute any one of the three variables, if we know the other two. If the arrival rate of the jobs is more than the capacity of the site to compute, then we export jobs to some other site which is less loaded and we can compute the results within less time than the current site. The exportation involves DIANA calculating the best site. This in fact keeps the whole Grid in a balanced form and no site is overloaded or underutilized.
- When a site is assigned too many jobs, it can try to send some of these to other sites [106] [107], which have more free resources or are processing fewer jobs than the local site, at that point in time. In this case, the jobs move from one site to another based on the cost criteria described in earlier sections of this chapter. The scheduler queries all the sites for their average load at that time and calculates the associated costs. The one with the minimum cost will be selected. Once a job has been submitted on a remote site, the site at which it arrives will not attempt to schedule it again on some other remote site (thus avoiding the situation in which a job cycles from one site to another). To each site we submit a number of jobs and a job reads an amount of data from a local database server, and then processes the data. If a site becomes loaded and jobs need to be scheduled on a remote site, the cost of their execution increases as the database server is no longer at the same site. If the amount of data to be transferred is too large or the speed of the network connections is too low, it might be better not to schedule jobs to remote sites but to schedule them for local execution.
- In bulk scheduling there is a time threshold and a job threshold. If the number of jobs submitted from a particular user increases beyond the job threshold then the priority of the jobs submitted above the threshold number is decreased and the jobs are migrated to a lower priority queue. In other words, with an increasing number of jobs, the priority of jobs from a particular user starts to decrease. Moreover, a time threshold is included to reduce the aging affect. With the passage of time, the priority of jobs in the lower priority queues is increased so that it can also have a chance of getting executed after a certain wait time. In other words, the more time a job has to wait the more its priority continues to increase. This is illustrated in Figure 4.12.

- While migrating the jobs, only low priority jobs are migrated to other sites since high priority jobs will get the chance to be executed on the current site. If low priority jobs are sent to other sites, there is the possibility that these jobs will again get a very low priority on the destination sites. To avoid such a situation a two pronged strategy has been adopted. Firstly, it is ensured that the target site has fewer jobs than the current execution site, otherwise jobs are not exported to the target site. Secondly, it is also checked that there are fewer high priority jobs in the target site as compared to jobs on the current execution site. This is checked from the queue management module of the jobs on each site and is not included in the cost calculation mechanism discussed earlier in this chapter. To further compensate the migrating job, the priority of the exported job is increased a step further so that it is compensated for the time it spent in the migration process and for fetching its associated data. The job migration algorithm is explained in section 4.7.

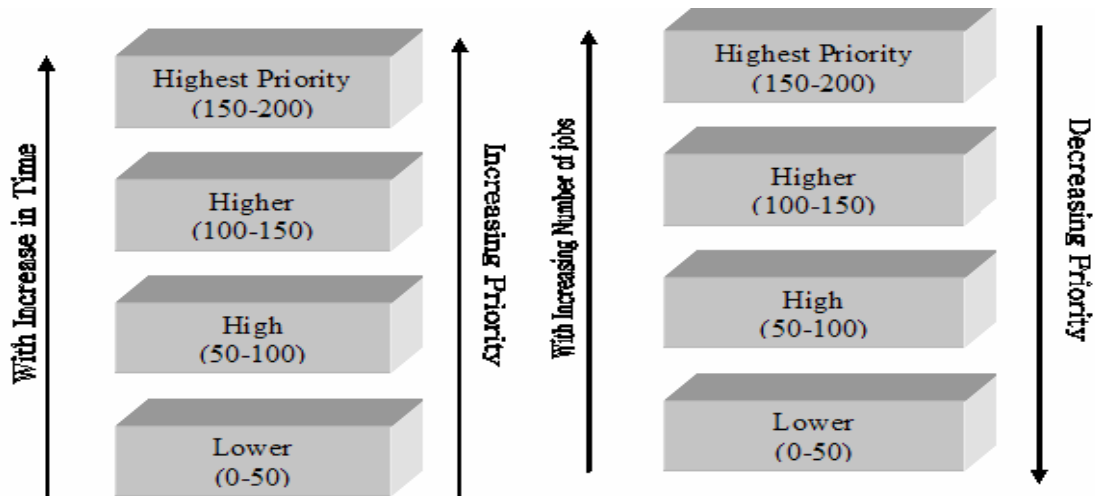


Figure 4.12: Priority with Time and Job Frequency

- If data is transferred to a site then it is permanently stored there, if storage space on that particular site permits this operation. This is not a big issue for the scheduling algorithm but could be a big issue for the overall performance of the system. The data stored on the target site is used for future jobs and this increases the overall efficiency of the system. But this results in the limitation that there should not be too many copies of the data around the world since an excess of storage copies will consume all of the spare storage space and this may restrict the future Grid operations. In conclusion, only the strategic placement and storage of duplicate data should be

allowed and only if a large number of arriving jobs may wish to use this data. Eventually this will enhance the efficiency of the system and, in addition, that of the algorithm as well.

- If the system obeys some quota and accounting based priority then a higher priority is assigned to jobs for a user having a larger quota and a lower priority for a small quota user. Again it is ensured that low priority jobs get the opportunity to execute and the priority of the jobs from the small quota user is increased after a threshold time limit. Similarly, the priority of jobs from the higher quota users should be reduced after the job threshold so that no such user can create starvation.
- Checkpointing [131] and process migration are important considerations if a pre-emptive scheduling policy is selected [130]. Due to the reasons stated earlier in this section, only non pre-emptive mode of scheduling is considered and hence checkpointing is not required in this case. Once a job has been assigned to a processor, the meta-scheduler can not pre-empt it until the job completes its execution. A job cannot start executing on a site as long as its required data does not become available and it must keep on waiting while its data is being transferred. This is one of the reasons for supporting non pre-emptive scheduling since data transfer costs can be very high in the case of data intensive jobs and check-pointing can be an expensive operation.

#### **4.6.4 Bulk Scheduling Algorithm**

The Bulk Scheduling algorithm works as follows. We take each bulk submission of jobs from a user as a single group. Each group is taken as a single job by the Meta-Scheduler which is scheduled by the DIANA algorithm, as discussed in section 4.4. If this group is too large to be handled by a site, it is divided into subgroups, each having a sizeable number of jobs which can be handled by any number of the sites in the Virtual Organization (VO). The VO administrator sets the size of the subgroups which are created if the size of the group is very large and cannot be accommodated by any single site. This size can differ from one VO to another. We assume that jobs are divided into equal but relatively smaller subgroups. The size of the subgroup is again set by the VO administrator. The size of the group is specified

in the job description language file since the user knows the number of jobs he enclosed in this bulk/group for his desired number of events/results.

First the scheduler checks whether the size of the group can be handled by a single site or not. Even if there is a site which can handle the whole group, it still checks whether it is cost effective to place this group on this particular site or whether it is more cost effective to divide the group into subgroups and submit the resulting subgroups to different sites. The group division factor mentioned in the algorithm below is a number used to divide the jobs into subgroups (see figure 4.13). The division factor varies according to the number of the jobs created from the splitting procedure and the capacity of the sites to execute these jobs. Smaller groups mean greater optimization since they can be fitted into a greater number of sites. Moreover, shorter jobs get higher priorities as discussed under DIANA Scheduling and therefore there are greater chances of their earlier execution and this improves the scheduling process. This also gives the advantage of including smaller sites into the execution process which otherwise will remain underutilized. While placing the group or its subgroups, the DIANA scheduling algorithm and the bulk scheduling routine discussed in Appendix 'A' are used and each group/subgroup is treated as a single job for the Meta-Scheduler. If the whole group is scheduled to a single site then the whole result is returned to the location which was specified by the user. In the case of subgroups, all the data from the subgroup execution sites is aggregated to a user specified location. No two groups from a single user or from different users can become part of a single group during the scheduling. Each group from each user maintains its identity and is treated independently by the scheduler. The pseudo code of the algorithm is given as follows:

```
Set the size of group in the jdl.  
Set the group division factor// division factor is equal to the number of subgroups as  
discussed above  
Submit the bulk Job in groups  
Get list of sites  
Check the queue size and computing capacity of each site  
Check the data location and data requirements of the group  
Match the site capacity against the bulk job group  
Use the DIANA scheduling approach to select a site  
  
If whole group can be accommodated by the site// site can execute the whole group  
    Submit the group to that site  
    Aggregate the output of all jobs in the group  
    return the results to the user's specified location
```

else

Divide the group into subgroups using the division factor  
Find the matching sites for the subgroups  
Submit each subgroup to different site using DIANA scheduling technique  
Aggregate the out put of all the subgroups  
return the results to the user's specified location.

All these values are specified in the job description language (jdl). jdl is being standardized by the GGF/OGF under Job Submission and Description Language (JSDL) but we will be using jdl (non-standard version) since EGEE and the CERN experiments are using it as a default submission and description language. However, use of JSDL is being considered for future developments. An example will be described in chapter 5 to explain the usage of jdl in the DIANA meta-scheduler. To prove the effectiveness of the algorithm, an example is given below. For example, the user submits 10,000 jobs in a bulk job. Let us suppose, there are four sites A, B, C and D having 100, 200, 400 and 600 CPU's respectively. We assume that the network and data conditions of all three sites are the same. Since these are bulk jobs, they have similar characteristics and we assume that each job in the group takes one hour to get processed. Using the algorithm stated above, we can have three possibilities: either to submit all the jobs on a single site, to divide the jobs into two best sites (here C and D) or to divide the jobs into four sites. Figure 4.13 gives the times taken in each execution process.

Jobs	Group division factor	A (100)	B (200)	C (400)	D (600)	Total execution Time (hours)
10,000	1				10,000	16.6
10,000	2			4000	6000	10
10,000	10	1000	2000	3000	4000	8.5

Fig 4.13: Job groups and execution improvements

From the table in Figure 4.13 we can see that by dividing the jobs into a number of groups the scheduler has clearly optimized the executions times. There can also be the possibility of a job execution limit on a site in which a user cannot execute more than a fixed number of

jobs. This concept of small groups will clearly help to optimize the scheduling process in this case. This allows a number of user jobs (bulk) to be grouped together as a single group job. If it is too large to be handled by any site then the scheduler divides it into sub groups and farms these out to all possible sites. But if it can be efficiently executed on a local site as against any other available site, DIANA meta-scheduler will schedule this group (or subgroups) to a local site. Furthermore there are certain large sites where, at a single point in time, all the processors might not be available (due, for example, to some prior executing jobs) and all the remaining available computing capability can be utilized by making small groups. This will reduce the queue as well as the load on large sites. This will also provide room for the high priority jobs to be executed since this will not starve a single site by allowing all the jobs in a group to be executed on a single site. However it does not mean that only computing power is taken into account as a submission criterion. Each group of jobs is submitted using the DIANA scheduling algorithm which ensures that only that site is selected for a group or a single job which has the least overall cost for its execution. We also described in the DIANA Scheduler algorithm that shortest job first execution reduces the execution times of all the jobs. This principle is also applicable here. In case of larger groups, the waiting times for the jobs will be more and hence it will affect the overall execution time. Small groups will spend less time in the queue by getting higher priorities and therefore overall execution time will be further reduced.

#### **4.7 Job Migration Algorithm**

Consider a scenario in which a user submits a job to a meta-scheduler which places the job in a queue. If the queue management algorithm (see section 4.5) of the meta-scheduler decides that this job should remain in the queue, it may have to wait some time before it gets scheduled or before migration to another site. The Queue Management Module of the meta-scheduler will ask the Scheduling Module to migrate this job. One important point to be noted here is that we want to locate the site where this job can be executed earliest. Consequently, our peer selection criterion is based on two things: a minimum queue length and a minimum cost to execute this job on the remote site. The meta-scheduler will communicate with its peers and will ask about their current queue length and the number of jobs ahead of this job. The site with the minimum queue length and minimum total cost is

considered as the best site to where the job can be migrated. The algorithm will work as follows:

```

Sites[] = GetPeerList( )
int count = Sites.length // total no of sites
int queueLength [ ] = Sites.length
int job_priority = getCurrentJobPriority(job); int jobsAhead[] = new int[ count ]
for ( i=1 to count )
    jobsAhead [i] = getJobsAhead( Sites[i] , job_priority )
end for
int minJobs = jobsAhead[1];
String peer="";
//find the peer with minimum jobsAhead
for( j=1 to count )
    if(minJobs > jobsAhead[j])
        minJobs = jobsAhead[j];
        peer = Sites[j];
end for
if ( peer's jobsAhead < localsite's jobsAhead) then
    increase the job's priority
    migrate the job to that site
else
    keep the job on local site

```

This algorithm works in the following manner. Firstly, the algorithm will get the information about the available peers from the discovery service. Then it will communicate with each peer and collect the peer's queue length, the total cost and number of jobs ahead in terms of job priority. It should be noted that the DIANA meta-scheduler follows the same queue management policies and characteristics across all the sites since the same meta-scheduler is installed at each site to interact with the local scheduler. The architectural details are discussed in chapter 5. Then it will determine the site with the minimum queue length and the minimum jobs ahead. If the number of jobs and the total cost of the remote site is higher than the local cost, then this job is scheduled to the local site (i.e. it will not be migrated). If other sites are congested then there is no benefit in migrating the job, and that job will remain in the local queue and will eventually be served on the local site. Otherwise, the job is moved to the remote site, subject to a cost mechanism. Note that the DIANA meta-scheduler does not consider each job for the export process, rather a group of jobs is exported to a remote site thereby significantly saving execution time on the remote site. It will not be cost effective to poll the remote peers and collect the queue and cost information for each job. This process is only carried out for bulk jobs or groups of jobs which are likely to take more

time on their local sites. Furthermore, this will reduce communication traffic between the peers since all peers are polled only after some intervals when jobs at the sites need to be exported. Otherwise if all peers are polled for each job, this would significantly increase the communication traffic between peers.

#### **4.8 Conclusion**

In this chapter, a theoretical as well as a mathematical description of the DIANA meta scheduling algorithm and bulk job scheduling has been outlined. It was shown, with the help of mathematical equations, that a matrix of different scheduling costs can significantly improve the scheduling process if each job is submitted and executed after taking into consideration certain associated costs. Queue time and site load, processing time, data transfer time, executable transfer time and results transfer time are the key elements for improving the scheduling (and execution) and these elements were represented in the DIANA scheduling algorithm. The three key costs which need to be calculated were identified as data transfer cost, compute cost and network cost and were expressed in the form of mathematical equations. It was established that if queue, priority and job migration was included in the DIANA scheduling algorithm, the same algorithm could be used for scheduling of bulk jobs. As a result, a multi-queue, priority-driven feedback based bulk scheduling and job migration algorithms which extend the DIANA scheduling algorithm were proposed and illustrated.

In the next chapter, the design and architecture details of the DIANA Scheduling system are discussed. The proposed system will implement the DIANA scheduling algorithms and will also help us to test the system through carefully created simulations. Design and architecture of the DIANA scheduling system and various associated scheduling hierarchies are discussed. Some aspects of the fault tolerance and the decentralized functioning of the schedulers are also outlined through a peer to peer approach.

## **Chapter 5**

### **Scheduling Hierarchies and DIANA Architecture**

Chapter 4 presented an overview of the DIANA scheduling system and the DIANA scheduling algorithms and discussed the details of the bulk scheduling process. Various scheduling approaches were elaborated for optimizing the data intensive scheduling process and a discussion of the issues related to bulk scheduling was made. Various scheduling costs were calculated and an analysis of the proposed scheduling algorithms was provided. In addition issues related to queue management, priority aware scheduling, starvation and aging were discussed.

Chapter 5 discusses the architecture of the Grid schedulers and related issues that optimize the scheduling and execution process. The issues which influence the scheduling hierarchies are elaborated in section 5.2 of the chapter. A comparison of different available scheduling architectures is presented in section 5.3. DIANA follows a P2P scheduling hierarchy and the reasons behind the adoption of this approach are stated in section 5.4. The DIANA scheduling architecture and scheduling API are presented in the section 5.5.

#### **5.1 Introduction**

The Grid concept was created to facilitate the use of available distributed resources effectively and efficiently. The first step needed before one can utilize the Grid for running jobs is to locate and use (the best) resources available to serve those jobs that is the process of resource scheduling. This chapter addresses the architectural and theoretical foundations of question 5 (described in chapter 1) which stresses that centralized algorithms and environments are less effective than their decentralized counterparts when scheduling data intensive bulk jobs. This chapter will also evaluate that applying the concept of P2P systems to resource scheduling can lead to efficient resource utilization.

A meta-scheduler coordinates the communication between multiple heterogeneous local schedulers that typically manage clusters in a LAN environment. In addition to providing a common entry point, a meta-scheduler also enables global access and coordination, whilst maintaining local control and ownership of resources through the local schedulers. The fundamental difference between a meta-scheduler and local schedulers is that a meta-

scheduler does not own the resources and has no autonomy in its decisions. Therefore, the meta-scheduler does not have total control over the resources. Furthermore, a meta-scheduler does not have control over the set of jobs already scheduled to a local scheduler (also referred to as the local resource management system). These local and meta-schedulers form a hierarchy and individual schedulers sit at different levels in the hierarchy as discussed by Mausolf in [141]. Each local scheduler can cooperate and communicate with its siblings through a meta-scheduler, however, each meta-scheduler cannot communicate with other meta-schedulers of other sites or Grids. Communication is only possible between local schedulers and the meta-scheduler. Existing scheduling systems are often based on the client-server architecture with one or several meta-schedulers on top of independent local schedulers such as LSF, PBS etc. Each local scheduler can collect information and can schedule the jobs within its own managed site. Typically, these local schedulers cannot schedule jobs to some other available site. Peer-to-Peer (P2P) scheduling systems on the other hand can provide environments where each peer can communicate with all other peers to make “global” decisions at each site, can propagate their information to other peers, and can control their behaviour through this information. In the P2P approach, a meta-scheduler and a local-scheduler make a hierarchy at each site where global decisions are managed by the meta-scheduler whereas local control and allocations are made by the local scheduler. The meta-scheduler on each site has access to global information and all meta-scheduler instances communicate with each other to share the cost and load information. Our intention is to incorporate a P2P approach so that schedulers do not take global decisions at a single central point, but rather many sites participate in the scheduling decisions through sharing the information in their cost matrices. Each site should have information on load, queue size etc., should monitor its processing nodes and then propagate this information to other peers. Local and certain global policies could be managed at the site level instead of a central hierarchical management. As a result, the P2P behaviour can become an important architectural model for fault tolerant, self-discoverable and autonomous global resource scheduling. This feature should make scheduling decisions more efficient. It is to be noted that MonALISA is the core provider of the peer-to-peer (P2P) behaviour and it inherits parts of the functionality from JINI. We selected MonALISA since it is the only monitoring tool which can provide the desired P2P features for DIANA. We use SOAP as well as XML-RPC for the communication. Further implementation details will be provided in chapters 6 and 7. In contrast to this approach, centralized scheduler management can be problematic in several

ways since load balancing, queue management, job allocation, policies etc. are central and are typically managed by a (single) central meta-scheduler and might not be fault tolerant. Note that by client server architecture, we do not mean here a tier system which uses various tiers, which are clients of each other, to scale up the client server behaviour. Each tier is not scaleable if treated in isolation.

Schedulers may be subject to failure or may not perform efficient scheduling when they are exposed to millions of jobs having different quality of service needs and different scheduling requirements. They may not be able to re-organize or export scheduled jobs which could result in large job queues and long execution delays. For example in High Energy Physics (HEP) analysis a user may submit a large number of jobs simultaneously (the so-called bulk job scheduling), and the scheduling requirements of bulk jobs may well be different to those of singly queued jobs. In bulk job submission by a single or multiple users at a particular site it might become impossible for a local scheduler to serve all the jobs without using some job export mechanism. In the absence of this mechanism, it is possible that some of the jobs might be lost by the scheduler due to timeouts before they get an execution slot, insufficient space in the queue to hold the jobs or the fact that the frequency of submission cannot be handled by the central scheduling site. What is required is a decentralized scheduling system which not only automatically exports jobs to its peers under potentially severe load conditions (such as with bulk jobs), but at the same time it manages its own scheduling policies, whilst queuing jobs and monitoring network conditions such as bandwidth, throughput and latency. The queuing mechanism that is needed at each scheduling peer should follow the same queue management scheme across all the sites in order to enforce uniform scheduling policies across the Grid sites. This will enable sites to interact and export jobs to other sites without any policy conflicts since all sites are following a similar scheduling approach. It should associate priorities to each job inside the queue, depending on the user profile and the job requirements with the scheduler servicing high priority jobs preferentially to optimise Grid service standards. In this chapter, we explain the functionality of a P2P meta-scheduler and present its scheduling and queue management mechanism and demonstrate the advantages and drawbacks of such a system implementation.

## **5.2 Considerations for Scheduling Hierarchies**

The following sections describe the considerations for selecting an appropriate meta scheduler hierarchy. The choice of a particular scheduling paradigm is dictated by certain criteria which are listed below. Whether a centralized, hierarchical or a distributed scheduling hierarchy is selected, each approach has drawbacks and certain advantages over the other approaches. We explain the issues associated with the selection of the scheduling paradigm and then provide details of various scheduling hierarchies in the next section on the basis of these criteria.

### **5.2.1 Performance**

When selecting a job scheduler and associated resources for the execution of the job, the Grid throughput and the performance requirements of the application must be considered. In most cases, an increase in Grid throughput will lead to a better performance of the applications and vice versa. The service requester is interested in a quality of service (QOS) that includes an acceptable turnaround time. While building a Grid and exposing one or more applications as a service over this Grid, the service provider would like to maximize the utilization of the resources and the throughput of the systems within the Grid to get a better return on the investment and to provide a better quality of service. Therefore a performant Grid is equally important for both the service requesters and the service providers. It becomes necessary for the scheduler to better decide the sites which can guarantee better performance. Hasher et al. [133] state that the order and hierarchy of the schedulers is critical for the selection of the better performing sites.

### **5.2.2 Reliability**

Availability of resources is a prime decision criterion for selecting a suitable scheduling hierarchy and it may have an influence on the scheduling optimization. The hierarchy should support the cases where the particular site or a scheduler becomes unavailable or where a scheduler might decide to migrate or to checkpoint a job. Resource availability is not only concerned with the choice of the sites and schedulers but it also deals with the execution software and environmental requirements for the jobs and the programs. When deciding the scheduling hierarchies, reliability should be the prime consideration so that the hierarchies

can deal with the host and network interruptions as well as being able to optimize the scheduling decisions. There are many approaches [134] that can be considered to make the schedulers and applications reliable, for example, using check point-restart mechanisms, having persistent storage to hold queue states in the scheduler and providing robust systems management solutions to maximize the availability of the Grid.

### **5.2.3 Scalability**

Modern day applications are not only compute and data intensive but also demand a high level of scalability. A system should be easily scaled to accommodate changes in the number of users, resources and computing entities affected by it. Scalability can be measured in three different dimensions: load scalability, geographic scalability and administrative scalability. A Grid system should support load scalability by making it easy for the users to expand and contract its resource pool to accommodate heavier or lighter loads. A geographically scalable system is one that maintains its usefulness and usability, regardless of how far apart its users or resources are. In administrative scalability, no matter how many different organizations need to share a single distributed system, it should still be easy to use and manage. Some loss of performance may occur in a system that allows itself to scale in one or more of these dimensions. There is a limit up to which we can scale/add processors to the system, and above which the performance of the system degrades. This feature should be considered in the scheduling system and the scheduling hierarchies should sense the network, load and other characteristics to make the scheduling decisions as described by Adami et al. [135]. Therefore before designing or enabling a Grid application, one should anticipate the scalability of the system.

### **5.2.4 Scheduling Policies**

A scheduling policy is a criterion which dictates the way the jobs are allocated to resources for execution. This is followed in order to allow certain users and processes to grant priority to others at certain points in time. The need for scheduling policies arises from the requirement to support different execution scenarios. For example, there is a scheduling policy that a job might not be started immediately and it might be interrupted or pre-empted during execution; for example it might be scheduled to run overnight. Another scheduling policy might be to allow only 50 percent resources for a particular VO with the remaining

resources being reserved for the local use. A job can be asked to run or to support a limited set of job categories or support a middleware which is installed on particular fire-walled hardware and this might be intended for scheduling and execution optimization [140]. Consequently scheduling policies can have a profound effect on the scheduling hierarchies in determining the QoS [136] and scheduling decisions for sending and retrieving jobs to and from Grid sites clearly depend on the scheduling policy for that site. The choice of the job policies and priorities is central to scheduling optimization; open execution of the jobs without any scheduling policy will deprive certain users from a sustained QoS and accordingly will alter the preferences in selecting a scheduler [138].

### **5.3 Scheduling Architectures**

Almost every job scheduling hierarchy has some benefits as well as limitations and these hierarchies are widely recognized with minimal variations. When submitting jobs for execution and scheduling and enabling applications for a Grid environment, the scheduling system needs to be aware of how the load-balancing mechanism (whether manual, push, pull, or some hybrid combination) will affect the application, specifically its performance and turnaround time. In this section we list major scheduling hierarchies and compare the pros and cons of each approach. This section will help explain the reasons which led us to select the P2P scheduling model.

#### **5.3.1 Master/Agent Architecture**

This is a centralized architecture for job schedulers and is suitable for the local or clustered scheduling. The Job Scheduling software is installed on a Master node while on the production machines only a very small component is installed (Agent) that awaits commands from the Master, executes them, and returns the results back to the Master. Condor [142], Sun Grid Engine [143], PBS [144] and others are based on this scheduling architecture. This is also known as centralized scheduling as shown in the Figure 5.1. In a centralized scheduling hierarchy all production machines are scheduled and managed by a central Master instance. Information on the state of all available systems is also collected by the Master node. The problem with this approach is that a centralized scheduling hierarchy cannot scale well with increasing size of the computational Grid. The central scheduler may prove to be a bottleneck in some situations e.g. if a network error cuts off the scheduler from its resources,

system availability and performance will be affected. As an advantage, the scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources.

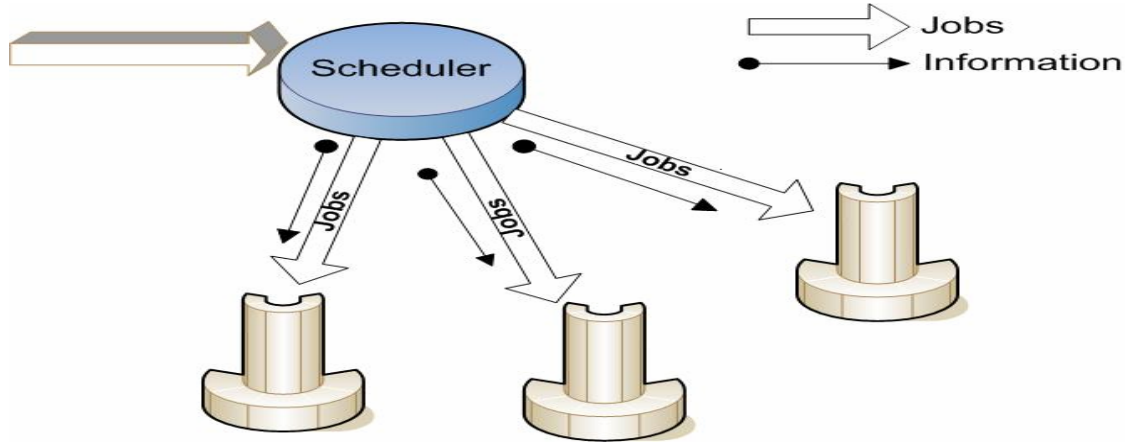


Figure 5.1: Centralized Scheduling Model

### 5.3.2 Push and Pull Model

When a job is submitted to a Grid scheduler, its workload can be distributed in a push model, pull model, or combined model. For example, the gLite workload management system implements both push and pull scheduling policies. A round-robin scheduler basically implements the push model of the scheduling. However, the push model does not consider the job queue lengths as discussed in [145] and forms a hierarchical architecture as shown in figure 5.2. In the pull model, synchronization and serialization of the job queue will be necessary to coordinate the pulling of jobs by multiple Grid resources [146]. Local and global job queues are also managed in the pull model. Failover conditions need to be considered in both of these scheduling models [147]. Therefore, the monitoring system should detect the non-operational Grid resources and no new work should be sent to failed resources in the push model. In addition, all the submitted jobs that did not complete their execution need to be taken care of in both the push and pull models. All the uncompleted jobs in the failed host need to be either redistributed or taken over by other operational hosts in the virtual organization.

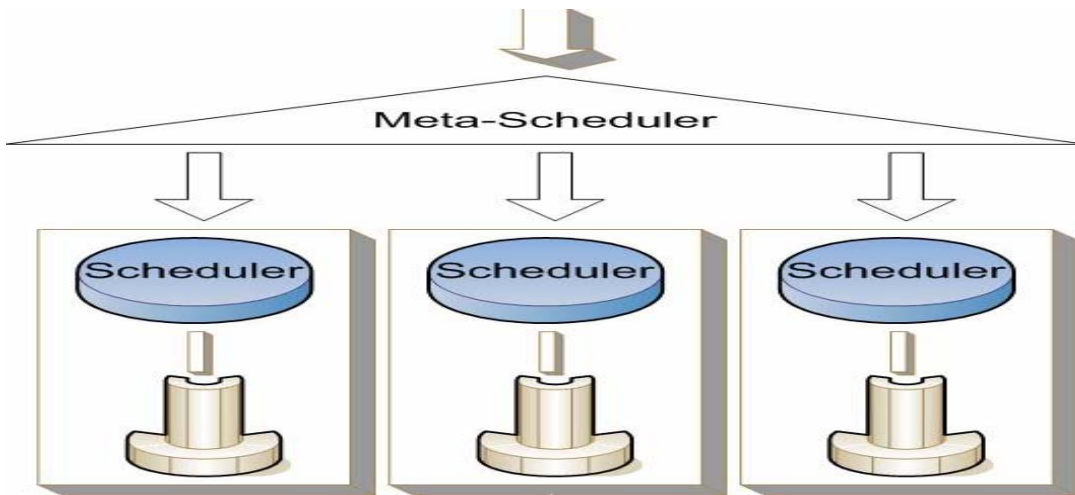


Figure 5.2: Push and Pull Scheduling Model

### 5.3.3 A Peer-to-Peer (P2P)-based Cooperative Scheduling Architecture

In a P2P scheduling model, each machine is capable of helping with the scheduling process and can offload locally scheduled jobs to other cooperating machines. In decentralized systems, distributed schedulers interact with each other and commit jobs to remote systems and no central instance is responsible for the job scheduling. Therefore, information about the state of all systems is not collected at a single point. Thus, the communication bottleneck of centralized scheduling is prevented which makes the system more scalable [148]. Also, the failure of a single component will not affect the whole scheduling system as is shown in Figure 5.3. This provides better fault-tolerance and reliability than is available for centralized systems without fall-back or high-availability solutions. But the lack of a global scheduler, which knows about all jobs and system information at every time instant, could lead to sub-optimal schedules. Nevertheless, different scheduling policies on the local sites are still possible. Furthermore, site-autonomy for scheduling can be achieved easily since the local schedulers can be focussed on the needs of the resource provider or the resource itself.

## 5.4 Hierarchies of Schedulers

In this section the DIANA peer to peer scheduling mechanism is discussed and its queue management process is elaborated upon. The peer to peer process to communicate and share the information between the meta-schedulers is illustrated and how they manage the jobs at a local and global level is highlighted.

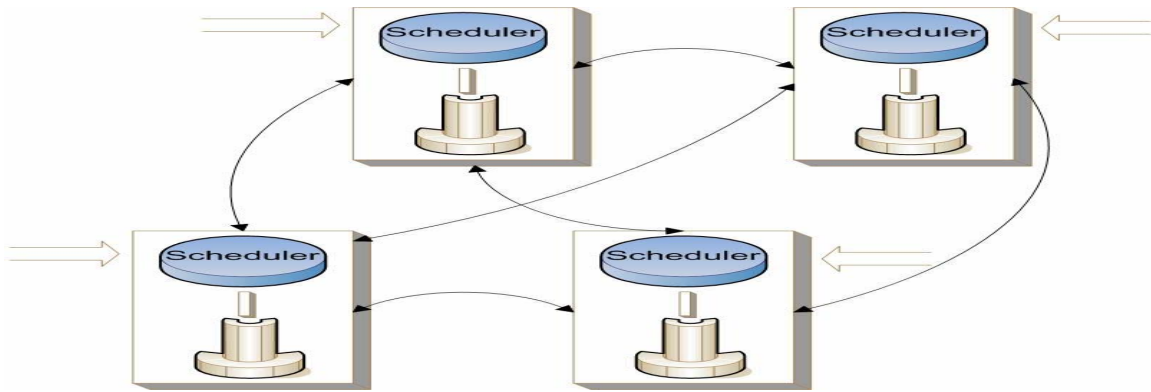


Figure 5.3: Decentralized Scheduling Model

A user submits a job to a meta-scheduler (local to the user, typically at the same *site*) which in turn contacts a local scheduler. A particular meta-scheduler considers only its own managed sites to schedule the job and does not look around for other sites managed by other schedulers to distribute load and to get the best available resources. The jobs are scheduled centrally irrespective of the fact that this may lead to a poor QoS due to potentially long queues and scheduling delays. Hence, the architecture with non-communicating meta-schedulers (see figure 5.4) can lead to inefficient usage of Grid resources. Furthermore, in this architecture the meta-scheduler schedules the job on its site, and it cannot communicate with the sibling meta-schedulers and hence does not consider the underlying network and data transfer costs between the sites. This is one of the reasons that almost all Grid deployments have at most only a few meta-schedulers and that any two cannot communicate and interoperate with each other.

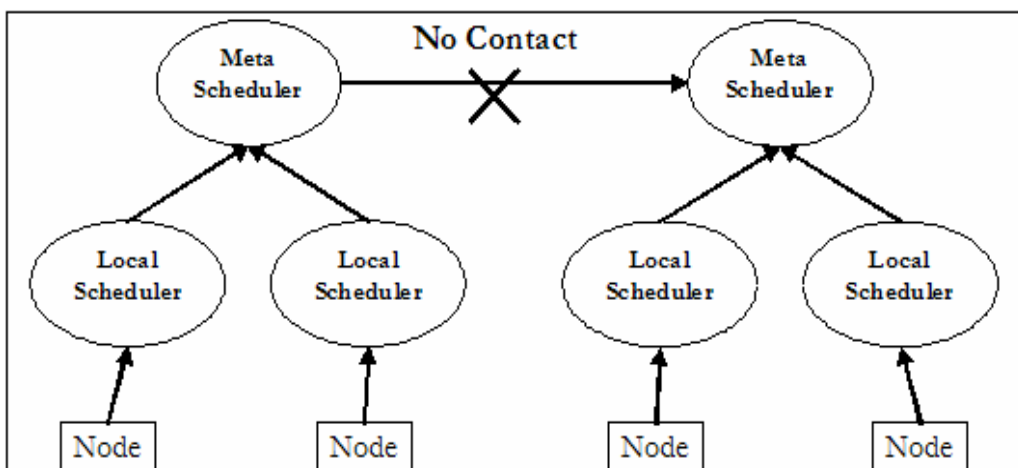


Figure 5.4: No Communication between Schedulers

Peer-to-Peer (P2P) systems, on the other hand, provide environments where each peer can communicate with all other peers to make the global decisions at a site level, can propagate its information to other peers, and can self-organize and control themselves using this information. This feature can make scheduling decisions and resource utilization more efficient [137]. Our intention is to incorporate a P2P approach for performing the scheduling process. Schedulers should not take global scheduling decisions at a single central point, rather all sites should participate in the scheduling decisions. Each scheduler at a site should monitor its resources and then propagate this information to other peers in the Grid. Each site should broadcast its load and queue size to other peers; local and global policies should be managed at the same level instead of a hierarchical management. As a result, this P2P behaviour can become a foundation stone for a self-managing, self-organizing and self-healing resource management and scheduling system. For DIANA a P2P decentralized and self-organizing scheduling system is required which cannot only automatically export jobs to its peers under severe load conditions, but it can also manage its own scheduling policies, hierarchy, queue and network conditions. Each peer should follow some queue management scheme which can associate priorities with each job inside the queue and can ensure Grid service standards.

#### **5.4.1 Meta-Scheduling with DIANA**

It is important in Grid systems to have a distributed meta-scheduler, which implements the features discussed in the previous sections, and that site meta-scheduler instances should interoperate and communicate with each other, should be fault tolerant and self-organizing and should make network aware data intensive decisions (including network characteristics in the scheduling decisions). In addition to being network-aware, the meta-scheduler should avoid making centralized decisions. It should communicate and share the information with all other meta-schedulers so that Grid resources are well evaluated and utilized.

DIANA is a Data Intensive and Network Aware meta-scheduler which performs global meta-scheduling in a local environment, typically in a LAN. In DIANA, we do not use independent meta-schedulers instead we use a set of meta-schedulers that work in a P2P manner. Each site has a meta-scheduler that can communicate with all other meta-schedulers on other sites as shown in Figure 5.5. The scheduler is able to discover other schedulers with the help of a P2P

discovery mechanism. We do not replace the local schedulers in this architecture rather we have added a layer over each local scheduler so that site meta-schedulers can talk directly to each other instead of getting directions from a central global meta-scheduler.

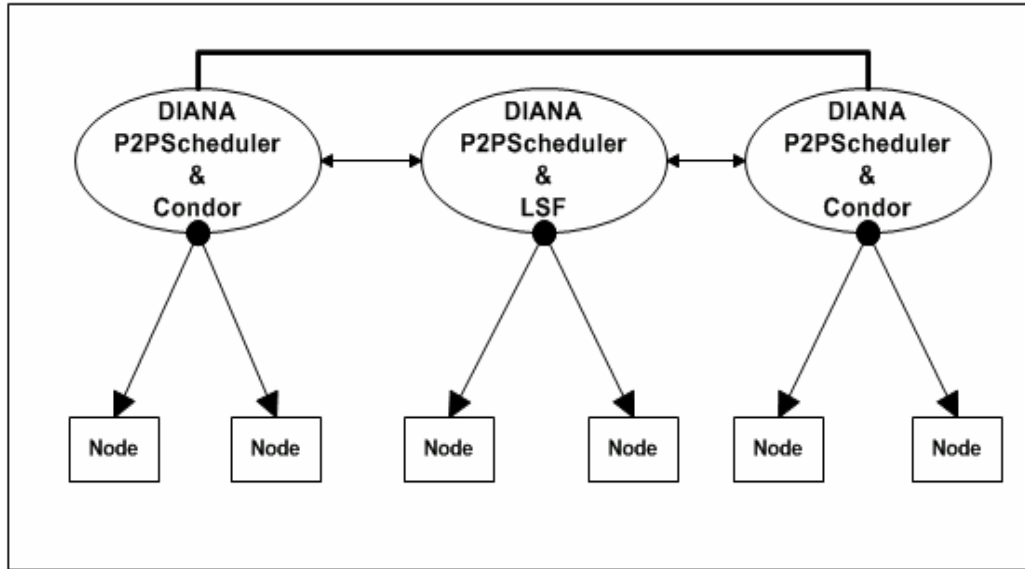


Figure 5.5: P2P Communication between Schedulers

A meta-scheduler can therefore obtain information from any other site and can make global decisions. Local information includes processing power, memory, site load, queue length and network capability. The meta-scheduler will make scheduling decisions based on three essential factors: the network cost, the computation cost and the data transfer cost. It can communicate with other meta-schedulers and may transfer jobs to other sites. It may transfer a job along with its required data to a remote site, consequently it should also consider the estimated transfer time of the job and data to that particular remote peer. Before making the scheduling decision, it should also consider the estimated computing capabilities of remote peers. Hence, the job will be submitted to the site with the least total cost.

In DIANA, the P2P behaviour is complemented by a discovery service (see chapter 6 for further details). This discovery service maintains a list of available/alive peers in different ways. One way is that whenever a peer meta-scheduler is introduced to the network, it will inform the discovery service about its availability and when a peer is properly shutdown, it will update the discovery service about its new status. This leads to the question: what would

happen if a peer suddenly went down without informing the discovery service? In order to cope with this issue, the discovery service uses an echo request/reply communication with the peers currently available in the list. The peer which does not reply is simply removed from the list. Each meta-scheduler site periodically contacts a discovery service to collect the updated information about the available peers. After getting this information, the peers start communicating with other meta-scheduling peers and update their local repositories with this information.

This approach is not simply an ‘all-to-all’ communication (or multicast in the network terminology) between the machines involved in the Grid system. The nodes are managed by local schedulers which report to the site meta-schedulers. The site to site communication is in essence a P2P communication between meta-schedulers. Each meta-scheduler maintains a table of entries about the status of the local schedulers, the queue length, jobs in execution mode, and the nodes managed by them which is updated in real time when a node joins or leaves the system. When a user submits a job, the site meta-scheduler communicates within the local scheduler to find the suitable resources. If the required resources are not available within the site, it contacts the meta-schedulers of other sites in the virtual organisation (VO) which have suitable resources. This approach is thus not just all-to-all communication and involves a reduced set of message passing between the meta-schedulers. In other words resources of each site are grouped under the site meta-scheduler and an overall system behaves like a group-to-group communication between the meta-schedulers. If each machine is allowed to communicate with other machines in the Grid, it can generate too much traffic, thus making the system less efficient. This approach not only provides a decentralized meta-scheduling environment but at the same time avoids network traffic issues. It also facilitates the overall Grid management by grouping the site resources under the respective meta-schedulers. Furthermore, communication between the meta-schedulers is not very frequent, meta-schedulers communicate only after fixed intervals to update the status of their resources to each other. A meta-scheduler might also require to communicate if a group of jobs at a site needs to be exported to a site having better resources. Therefore, this meta-scheduler communicates with other meta-schedulers for load evaluation and cost determination for job submission to that remote site.

### 5.4.2 Global Queue Management

In conventional client-server scheduling architectures, local schedulers handle their queues at the site level whereas a meta-scheduler has a global queue at some central location. However, in the DIANA architecture, there is one DIANA meta-scheduler at each site, i.e. the DIANA P2P meta-scheduler layer sits on top of one or many local schedulers at each site. In client-server architecture such as the one used by the gLite meta-scheduler, there is only one large queue at the meta-scheduler with local queues at each site. However, in the proposed P2P architecture each site meta-scheduler has knowledge about the local queue(s) plus a global queue which is managed by the DIANA layer. This leads to a scalable and self-organizing meta-scheduling behaviour which was previously missing in some of the conventional client-server scheduling architectures. Each meta-scheduler has a queue management mechanism where it can queue the incoming jobs in a Scheduler Queue as shown in Figure 5.6, and the meta-scheduler assigns priorities to the incoming jobs. In Grid scheduling we have “user quotas” (user quota is the number of jobs a user can submit within a definite period of time), network characteristics, data locations and securely granted user privileges and therefore, each meta-scheduler needs to maintain its queue according to these criteria.

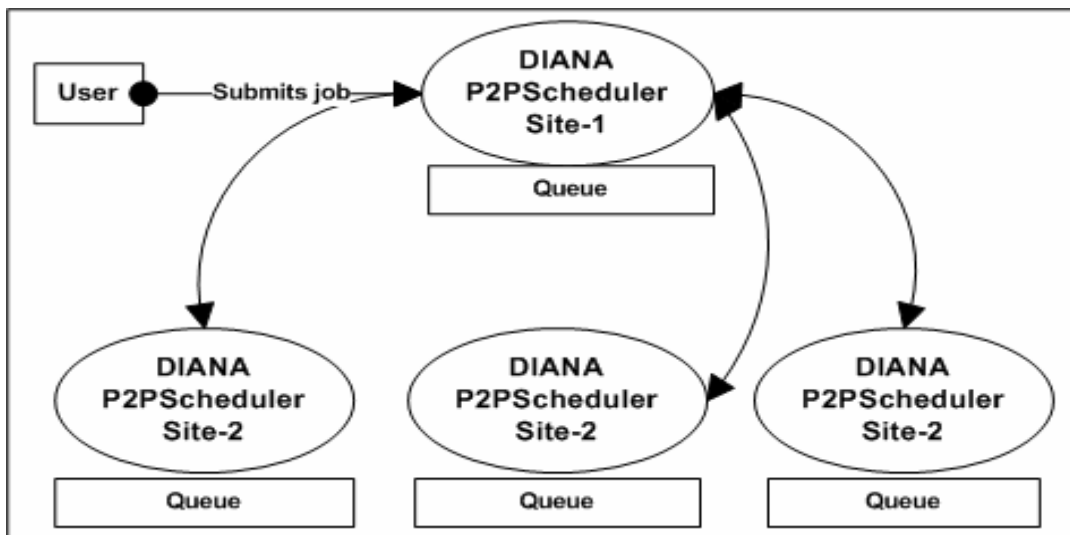


Figure 5.6: Queue and the DIANA Scheduler Instances

Queue management also facilitates load balancing and job migration. Before migrating a job, questions need to be answered such as: “What is the queue length on the target site?” “Can the target site execute the job quicker than the current site?” “If the job is migrated to another

site, what will be the job priority on the remote site?” “How many jobs are ahead of this job in terms of priority?” These considerations can have a significant effect on Grid performance. Figure 5.7 illustrates this queue management issue in the DIANA meta-scheduler. At each site there are two queues. One is the meta-scheduler queue and the other is the queue of the local or site scheduler. The meta-scheduler queue deals with the global jobs and takes into account the Grid information whereas the local queue is site specific. Jobs cannot be migrated if the meta-scheduler has scheduled them to any local scheduler and they will have to wait in the local scheduler queue until they get the execution slot on that site. Only the jobs from the DIANA meta-scheduler queue are exported to other sites. In contrast, once a job is allocated to a local scheduler at a site, it is never exported and waits in the local queue until assigned to a processor. All the prioritization of jobs, policy enforcement, migration and job steering issues are handled at the DIANA P2P level whereas the local scheduler works exactly in the same fashion as before once the job has been allocated to it.

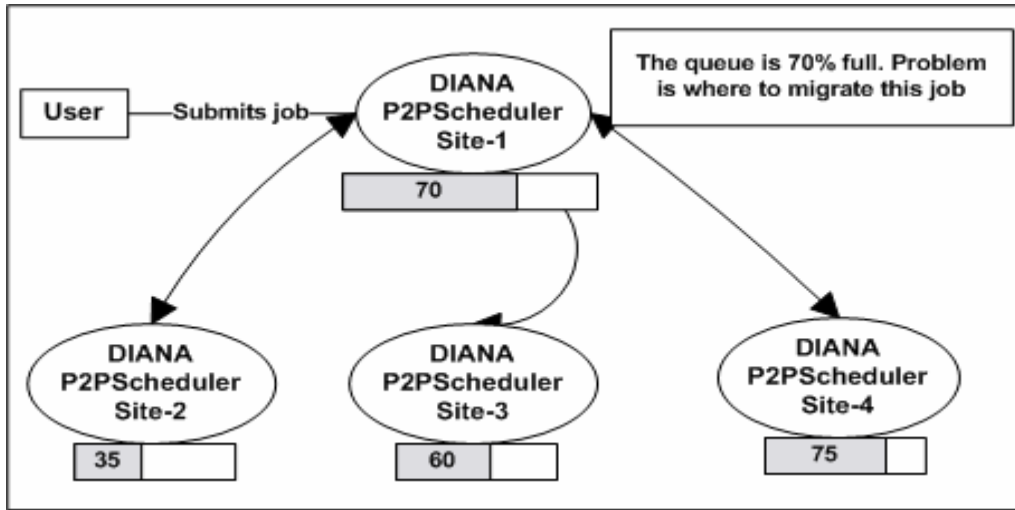


Figure 5.7: Queue Management in DIANA P2P Scheduler

## 5.5 DIANA Architecture

In the following sections the DIANA job scheduling architecture is described, to indicate where the previously introduced optimization algorithm and hierarchies can be applied. The DIANA Scheduler is introduced and some related services are discussed that can be used by a Grid job submission service for selecting a suitable execution site for a job.

### 5.5.1 General Architecture

The overall architecture of the DIANA meta-scheduler is shown in Figure 5.8. It includes a matchmaking layer which is responsible for selecting the best resources for the job's execution. The matchmaker is a component of the meta-scheduler but it can also work with other meta-schedulers, for example the EGEE meta-scheduler. The matchmaker uses the network characteristics provided by the network monitoring service, an optimized replica provided by the Data Location Service and other information services to make optimal scheduling decisions. The implementation details are given in chapters 6 and 7 but here we clarify that the underlying protocol for communication between these services is SOAP. The Data Location Service makes use of the Data Location Interface [149] to find the list of the dataset replicas and then uses network characteristics to find the "best" replica which is then chosen by the scheduler. The scheduler provides coordinated access to the underlying resources of a Virtual Organization (VO), regardless of their physical location or access mechanisms. When an application using a Grid makes use of more than one physical resource during its execution, the Scheduler maps the resource requirements to the multiple physical resources that are available to run that application. The meta-scheduler is the key to making the VO resources easily accessible to end-users, by automatically matching the requirements of a Grid application with the available resources while staying within the conditions that the VO has specified with the underlying resource managers.

The meta-scheduler (see Figure 5.8) is used for job management and execution including allocating resources needed for any specific job, the partitioning of jobs to schedule parallel execution of tasks on local schedulers, data management and the service-level management capabilities. Several schedulers form a hierarchical structure, with meta-schedulers forming the root of the hierarchy. Other lower level schedulers (that are part of the Local Resource Management Systems) provide specific scheduling capabilities that form the leaves. These lower level schedulers are constructed with a local scheduler implementation approach for specific job execution, or another meta-scheduler or a cluster scheduler for parallel executions. There can even be a meta-scheduler at the local resource management level; this is the case when a site has a large number of resources which in turn are managed by different local resource management systems.

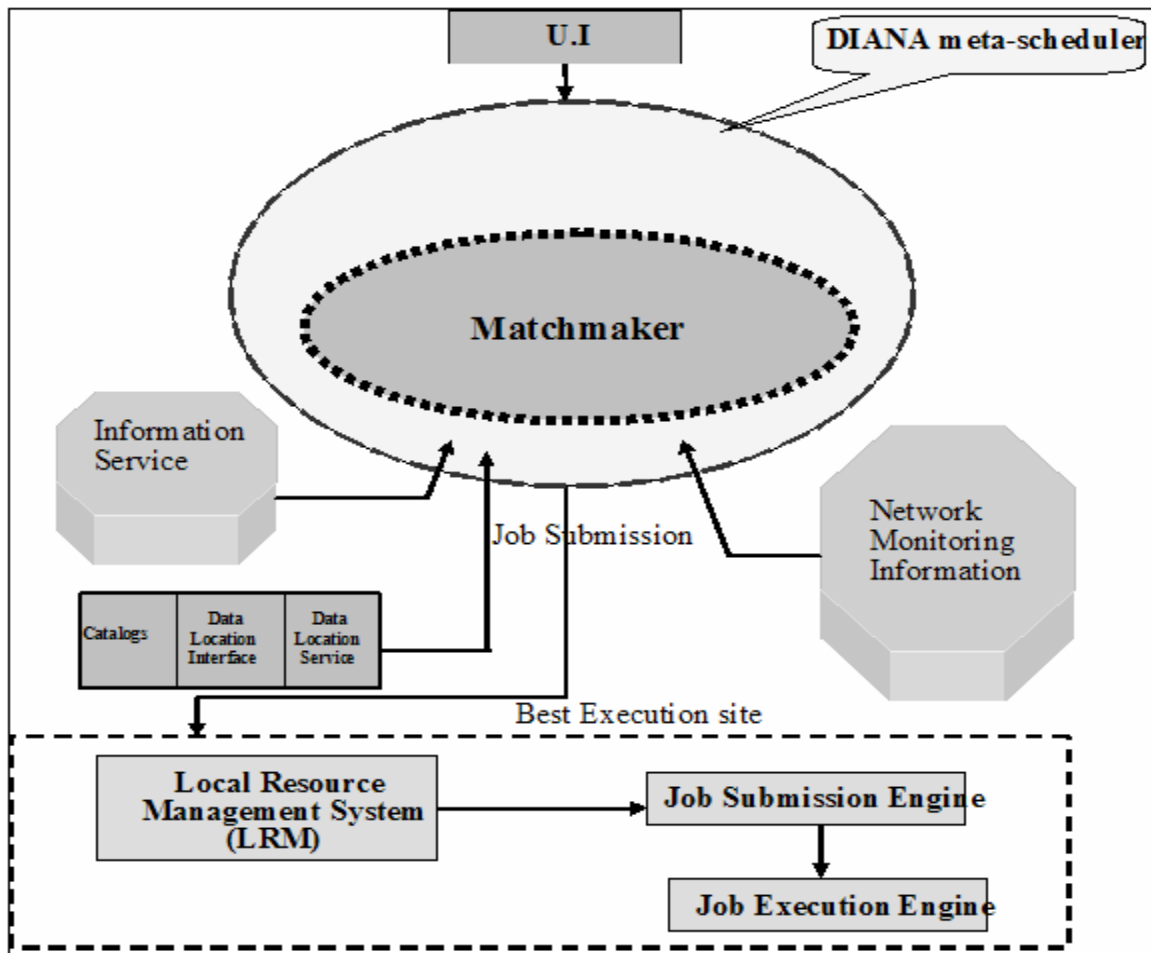


Figure 5.8: A generic job submission architecture with the DIANA Scheduler and the Data Location Service that are used for scheduling data intensive jobs.

The next important task is to select and match the appropriate resources for job execution: matchmaking. A particular matchmaker provides feedback to the users on the available resources. In general cases, the matchmaker may select a suitable scheduler for the resource execution task, and collaborate with that scheduler to execute the task(s). The pairing enables the selection of the best available resources from the service provider for the execution of a specific task. The matchmaker collects information (such as resource availability, usage models, capabilities, and pricing information) from the respective resources, and uses this information source in the pairing process. The matchmaker uses the cost matrix to select the appropriate resources for the job execution. A meta-scheduler serves each VO and end-users submit their Grid jobs to the meta-scheduler, which in turn matches the resource requirements of the jobs with the underlying physical resources through interaction with local

resource managers. The meta-scheduler uses the matchmaking engine, information services and monitoring information to decide the choice of a particular site. Data related information is provided by the Data Location Interface from various catalogues which is then optimized by the Data Location Service.

Users specify the requirements in the form of a Job Description Language (JDL) which is parsed by the matchmaker and respective services are queried to find the information about the resources which match these requirements. The DIANA Scheduler service mediates between the providers and the requesters and uses the scheduling algorithm of chapter 4 to select the best resource. The first step is to discover the resources. A resource request, simply called a request, consists of a function to be evaluated in the context of a resource. For example, the request “processing power > 2 GHz” will be evaluated by determining if a resource has an attribute called processing power and if so whether the value of this attribute satisfies the condition “Value(processing power) > 2 GHz”. If the request can be fully satisfied, the matchmaking service responds with a list of ranked resources. After this, we use the scheduling optimization algorithm to select the best resource and subsequently a job is scheduled to be executed on this resource.

Load-balancing is the distribution of workload among the resources in a Grid environment and this is managed by the scheduling system. This load-balancing also helps the meta-scheduler to avoid processing delays and over commitment of resources. The DIANA Scheduler keeps track of the load on the sites through the P2P mechanism stated earlier and selects a best site using the criteria discussed in chapter 4. Network monitoring information is the central component of the system and all the information collected is stored in a database and is used to make scheduling decisions. It collects the network information after specified intervals to get idea of the system state for improved decision making and its explanation is given in chapter 6. The exact interval to collect the network information can be changed by the site administrator through changing the value in the configuration file but we are using a 15 minute interval for our experiments. One can have longer intervals, for example 30 minutes, but such old information might not be very useful for accurate scheduling decisions since Grid resources are dynamic and their status can change quickly. Using information which is 30 minutes old might not give accurate schedules. However, on the other hand, if there are too short intervals, for example one minute, this can generate too much traffic

especially when the information is being shared between all the peers. There is an execution service which submits the job to the site selected by the DIANA scheduler. This also gets the information of the job state during execution and reports the progress as well as providing job completion information and stores these parameters in the database for later access by the users.

### 5.5.2 DIANA's Scheduler Interface

DIANA's scheduling interface is an API through which an application program accesses DIANA's matchmaking services. This scheduling API is defined at source code level and provides a level of abstraction between the applications and the DIANA scheduler. The job submission and job control services use this API to submit and execute jobs to one or more sites. The objective is to facilitate the direct interfacing of applications to the DIANA scheduler and provide a generalized API to facilitate integration of application programs. The scope of this API is limited to an appropriate CE and SE selection for the data intensive jobs. Job submission, job monitoring and control, and retrieval of the finished job status are managed by the special customized services and are not exposed through this API. An end user that interacts with a Grid job submission service uses the Job Description Language (JDL) [150] to specify certain requirements that need to be parsed and analyzed by the job submission service. The DIANA Scheduler provides a simple interface for this job submission service for selecting a suitable Computing Element (CE) for a given data intensive job. The JDL also helps to define the requirements from where the data will be read (defined as *inputData* in the discussion below) and where output data needs to be stored. The name of the logical dataset, any programs required for execution and any computing and storage requirements are specified in the JDL which is then passed to the DIANA Scheduler since the meta-scheduler calls the DIANA Scheduler for the match-making process. These requirements are passed to appropriate services (such as the Data Location Service, described in chapter 7) for decision making. The following is a sample of JDL which will be used by the Meta-Scheduler and appropriate services of the DIANA Scheduler which will be called for decision making.

```
VirtualOrganisation = "cms";  
Executable = "PhysicsApplication";  
InputData = {"lfn:physicsFile1", "lfn:physicsFile2", "lfn:physicsFile3", "lfn:physicsFile4" };
```

```


DataAccessProtocol = {"root", "Gridftp"};
output_file = output.root, output.txt, FrameworkJobReport.xml
additional_input_files = mydata.dat, input.root, input.txt
copy_data = 1
storage_element = srm.cern.ch
storage_path = /srm/managerv1?SFN=/castor/cern.ch/user/u/username/subdir
//output file specified by output_file will be placed gsiftp://storage_elementstorage_path/output_file_job#.ext
register_data = 1
lfn_dir = myusername/subdir
lcg_catalog_type = lfc
lfc_host = lfc-cms-test.cern.ch
lfc_home = /grid/cms


```

In the push based scheduling model which DIANA is following, the Grid scheduler sends the resource requirements of the job to its peers. Once the job has been submitted to the DIANA scheduler, the matchmaker needs to find a suitable site where the job can be executed. The matchmaker first needs to parse the JDL and find resources that match the job requirements defined in the JSDL. The available Computing Elements are obtained from the Discovery and Information System discussed in chapter 6. In case of data intensive jobs that require InputData, the matchmaker needs to contact the Data Location Service in order to retrieve Storage Elements where physical files are located. In detail, the physical file location is identified by an URL that contains the hostname of a Storage Element (SE). Once the matchmaker has retrieved the list of all SEs, a site (Computing Element) needs to be selected that is “close” to a given SE. The term “close” refers to closeness in terms of network connectivity, i.e. a CE is close to an SE if it is available in the same local area network or has the least network cost. If several CEs satisfy the job requirements, the scheduler selects the best of them and finally sends the job there.

To sum up, resource selection of data intensive jobs is based on optimising compute and data transfer times, i.e. a job is sent to a Computing Element that in turn is close to a Storage Element which contains the necessary data or data is sent to a Computing Element which in turn selects the best Storage Element. Before a job begins to execute, it needs to wait in a site queue before the required data becomes available on that site. Network cost determines the approximate data transfer time from initial node to destination node. The scheduler decides

the destination system for the job which is selected based on the overall cost, which includes compute, network and data transfer costs. The job is then sent to the scheduler of the destination site according to the availability of resources and the requirements of the job.

From the example JDL it is required that a program called PhysicsApplication is available. Next, the program requires a physical replica of a file on the CE where the job is going to be executed and this step may involve replication of the required data. Job run-time takes place on a Worker Node (WN) of a specific Computing Element (CE)/local resource management. Jobs arrive on the WN with an application configuration which is site-independent. The CE/WN is expected to be configured such that the job can determine the locations of necessary site-local services (local file replica catalogue, CMS software installation on the CE, access to CMS conditions, etc.). Input data requirements are specified by logical filenames which are used to return the physical replica. The protocol used to access the physical replica of the dataset has to be either root (the access protocol for the ROOT object storage system [151]) or Gridftp and this is specified in the DataAccessProtocol. If the job being specified through the JDL is a bulk job then after parsing the JDL, the job is sent to a splitting module. There are number of applications available which perform the splitting and combination procedure, for example the CRAB utility [165]. After splitting procedure, the jobs are sent to the DIANA queues which are then scheduled using the scheduling mechanism discussed before. In order to select a certain CE for a given job with its data requirements, the DIANA Scheduler utilizes two internal methods that work in the following manner:

- A suitable CE is selected on the basis of computation, network and data transfer costs. The ultimate destination of the output data is also taken into account for selecting the best CE. This functionality is implemented in the DIANA scheduler part of the architecture.
- For the given inputData and CE find the “best” Storage Element (SE), where best refers to the minimal data transfer and network costs. The actual implementation of this functionality is done by the Data Location Service (see chapter 7).

In more detail, the two methods look like:

- String GetBestComputingElement ()

This method returns the best CE with respect to job requirements which are passed by the meta-scheduler or directly by some other client. This method takes into consideration the number of processors at a site, the load and queue size and the distance from the submission site or from the location where the output data is required. This method gets the information about the CEs and the load from the Information System. After ranking them it selects the CE that has lowest cost which is then passed to the *GetBestStorageElement* method below to find an optimal physical replica with respect to that CE. This method uses the network cost to decide the relative cost of the output data location and uses the computation cost to rank the CE. A combination of both of these costs helps to select a best CE.

- String GetBestStorageElement (String inputDataType , String inputData , String BestComputingElement)

This is used to find the best replica of a dataset. The caller of the method provides the logical dataset name (*inputData*) and its type (*inputDataType*), and the best physical replica of that logical dataset name is returned. These replicas are ranked with respect to a CE which is selected in the *GetBestComputingElement* method. This CE is passed as a third parameter (BestComputingElement) to this method. Again, this method can be called either by the Meta-Scheduler or by any other client to find the best physical replica of a dataset with respect to a certain location. This method uses the network and data transfer costs as calculated in the scheduling optimization algorithm to find the best physical replica of a dataset which will be used as an input to the job.

## 5.6 Conclusions

In this chapter, we described the scheduling hierarchies and elaborated the factors which can influence the scheduling architectures. Various scheduling hierarchies were discussed and it was concluded that centralized scheduling models are not adequate for complex scheduling scenarios as in the case of bulk scheduling. We discussed how a P2P scheduling model is ideal for the DIANA Scheduling and presented its queue management mechanism. We then described the DIANA scheduling architecture, its load balancing mechanism and illustrated the DIANA scheduling API.

Chapter 6 describes the process of extracting and analyzing the network values and their impact on scheduling decisions. It also explains the role of Discovery and Information

Services in the Meta-scheduling and how this facilitates the P2P scheduling process. This chapter also discusses monitoring tools and information services for discovering the services, for measuring the network parameters and for collecting the required network data.

## **Chapter 6**

### **Network Aware Meta-Scheduling**

Chapter 5 explained the architecture of the DIANA scheduling and the merits of P2P over decentralized scheduling were detailed for job scheduling. The building blocks of the P2P scheduler were explained and the issues to tackle priorities and policies in the DIANA Scheduler were highlighted. Various scheduling topologies and hierarchies were illustrated and their association with DIANA meta-scheduling was discussed. Queue Management and scheduler interfaces for the DIANA scheduler were also covered.

Chapter 6 describes the process of extracting and analyzing the network values and their impact on scheduling decisions. It also explains the role of improved network performance in meta-scheduling and how network managed services can influence data intensive scheduling decisions. This chapter also discusses monitoring tools and information utilities for discovering the services, for measuring the network parameters and for collecting the required network data.

#### **6.1 Introduction**

Grid applications are increasingly becoming dependent on efficient network support, but these applications often do not take full advantage of the new high-speed networks which are becoming pervasive. This is largely due to the fact that the applications use the default network characteristics which were not specifically designed for Grid use [152]. Grid applications cannot change their behaviour with a change in the network conditions as is the case with network aware applications that respond according to the network weather. On the other hand, Grid applications are massively distributed across the global Grid network; application instances are shared and they need to communicate across the geographical boundaries. They exchange large amounts of data and compute resources are shared between a single or multiple applications, resources are dynamic and may come and go at any time; some network links are stable while others are congested and unreliable. All these factors make Grid applications unique in the sense that they heavily depend on the underlying network and should be capable of responding to network needs and characteristics. The consequence of not adopting the network characteristics in the applications has been to sacrifice optimal Grid throughput in exchange for fair sharing of bandwidth and other

network resources on congested networks. This becomes further complex if data intensive applications are being scheduled and executed since such applications can consume and produce a lot of data. In order to overcome this limitation, Grid applications running over high-speed wide-area networks need to become “network-aware” [13][14], which means that they need to include the networking parameters in the scheduling decisions and adjust resource demands to the current network conditions.

What is really required by these applications is the creation of a next generation network-aware Grid middleware, which is able to effectively manage the network resource in terms of scheduling, access and use [155]. Conversely, the specific requirements of Grid applications provide drivers for research towards the development of Grid-aware networks. Cooperation between Grid middleware and network infrastructure is a key factor in effectively enabling the execution of network-intensive applications that require massive data transfers, very fast and low-latency connections and stable and guaranteed transmission rates. Large e-science projects, as well as industrial and engineering applications for data analysis, image processing, multimedia, or visualisation amongst others await efficient Grid network support. A network aware and network managed Grid middleware will enable end-to-end dynamic bandwidth allocation and low-latency access, inter-domain access control and network performance monitoring capabilities.

Today’s data intensive sciences, such as High Energy Physics (HEP), need to share data at high speed. This in turn requires high-performance, reliable end-to-end paths between the major collaborating sites [156]. In addition end-users have long and short term expectations for network and application performance for planning and trouble-shooting. To enable this requires a network monitoring infrastructure which is able to provide measurements and analysis of network performance between the major sites. Therefore for Grid scheduling and execution decisions, a robust performance monitoring infrastructure is required which can collect and feed this information to the scheduler. Moreover, network performance monitoring is important to the operation of Grid networks of any significant size as an aid to fault detection and for determining expected performance and its inclusion in the scheduling decisions can prove to be an important factor for meta-scheduling optimization. The need for such monitoring is enhanced in the Grid scheduling [157] arena due to the following reasons:

- The Grid scheduler can optimize its performance by selecting the *best* resources for compute and data intensive jobs, and adapting to changing network conditions for execution performance gains.
- The Grid scheduler relies on network efficiency, as an essential step in supporting data intensive applications, since simple over-provisioning of the network is not desirable for job scheduling on any particular site. Over-provisioning is commonly used to protect network performance against traffic variations, be they caused by failures or transient surges. However, the emergence of high-speed applications and access links, and the ever greater heterogeneity of user traffic profiles, e.g. machine-to-machine mean that there is considerable uncertainty regarding whether such conservative over-provisioning factors [166] can even remain adequate as networks and the user population they serve continue to grow.
- Network information can make the scheduler self-healing and help support adaptive decision making in the scheduling. This behaviour is particularly dependent on the network information.

## 6.2 Network Measurements and Meta-Scheduling

The meta-scheduler can enhance its performance by collecting and using information on the status and behaviour of the underlying infrastructure. Various network metrics can be made available to the scheduler so that a comprehensive and up-to-date picture of the Grid network status can be constructed. Network metrics can be used to optimize scheduling and for dynamic adaptation in a number of scenarios. The three terms used most often to refer to the overall performance of a network are speed, bandwidth, and throughput. These are related and often used interchangeably, but are not identical. The term speed is the most generic and often refers to the rated or nominal speed of a networking technology. Bandwidth can mean either a frequency band used by a technology or more generally to data capacity, where it is more of a theoretical measure.

Throughput is a specific measure of how much data flows over a channel in a given period of time and is usually a practical measurement. One very important, but often overlooked term for job scheduling is the timing of data transfers on a communications channel or network. An important aspect of latency is how long it takes from the time a request for data is made

until the data starts to arrive. Another aspect is how much control a device has over the timing of the data that is sent, and whether the network can be arranged to allow for the consistent delivery of data over a period of time. Data Transfer time and location are thus two important parameters for meta-scheduling.

In a Grid environment, network monitoring is vital in determining the source of performance problems or to tune the system for improved performance. Consequently, network performance data is crucial to Grid scheduling especially in the case of data intensive scheduling. Accurate and up-to-date network information can lead the scheduler to significantly optimize the job execution process. The following are the network related parameters [158] which were discussed in chapter 4 and which are important for data intensive scheduling decisions:

- Delay: the time taken for data to reach its destination and back. This is also called the round trip time.
- Network bandwidth: the amount of data it is possible to send between two sites per second.
- The route taken by the data across the network domains.
- Flow-monitoring: the ability to track the data traffic flows originating from a specific source. Both the route taken and traffic flow effects are sources of jitter and delay in a network. For example, flow-monitoring can help to check peering. Peering requires the exchange and updating of router information between the peered networks or ISPs, typically using the Border Gateway Protocol (BGP) which not only can reduce the delays but can also significantly influence the data transfer time.
- Reliability and consistency of the network, which is measured through the packet loss.

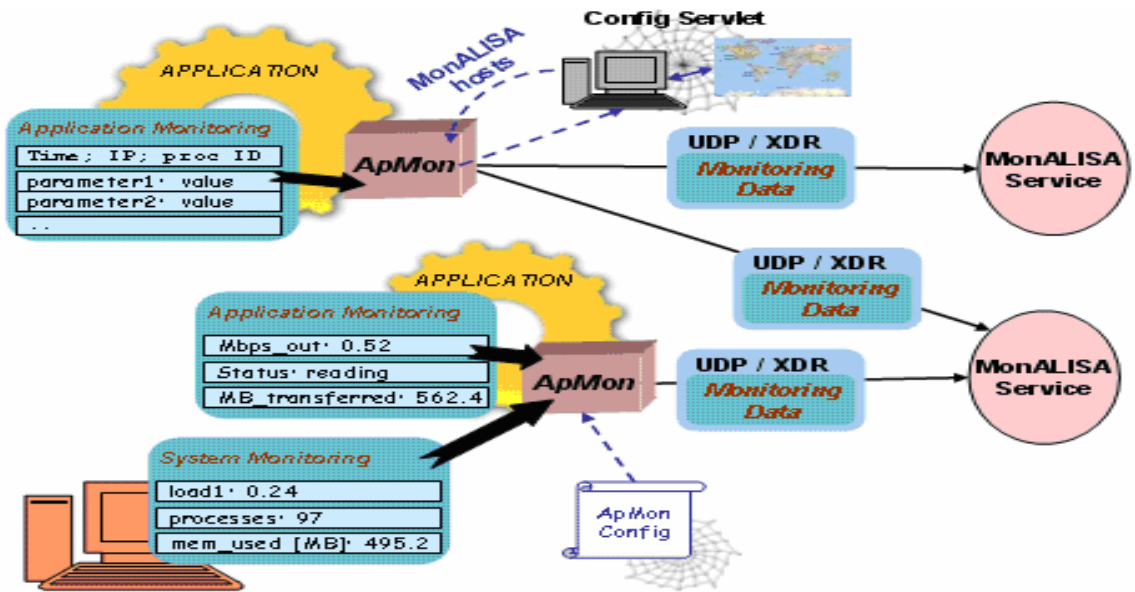


Figure 6.1: Monitoring data propagation through MonALISA (adopted from <http://MonALISA.cacr.caltech.edu>)

### 6.3 Monitoring Infrastructure for the DIANA Scheduling

We selected two monitoring tools to collect the network data which we will be using in the DIANA Scheduling process. One is PingER [12], a monitoring tool developed at SLAC, and the other is a JClarens based information and discovery service [160] that uses a MonALISA repository to access the network information. JClarens [161] uses the global information system provided by MonALISA [11] for its discovery and monitoring services. The service oriented architecture of MonALISA enables others to integrate different kinds of monitoring tools like ganglia and SNMP which can collect information describing performance parameters such as network and application performance indicators. As shown in Figure 6.1, arbitrary monitoring data can be published to a MonALISA station server using Application Monitoring API (ApMon), a library that uses simple External Data Representation (XDR) encoded UDP packets. ApMon is a set of flexible APIs that can be used by any application to send monitoring information to MonALISA services. The monitoring data is sent as UDP datagrams to one or more hosts running MonALISA services. Applications can periodically report any type of information that the user wants to collect, monitor or use in the MonALISA framework to trigger alarms or activate decision agents. JClarens uses the ApMon library to publish monitoring information to the MonALISA JINI network. The JINI

architecture [167] federates groups of devices and software components into a single, dynamic distributed system. JINI provides the notion of service lease time and enables services to find each other on a network and allows these services to participate and cooperate within certain types of operations, whilst interacting autonomously with clients or other services. This architecture simplifies the construction, operation and administration of complex systems: (1) by allowing registered services to interact in a dynamic and robust (multithreaded) way; (2) by allowing the system to adapt when devices or services are added or removed, with no user intervention and (3) by providing mechanisms for services to register and describe themselves, so that services can intercommunicate and use other services without prior knowledge of the services' detailed implementation. Each MonALISA service registers with a set of JINI Lookup Discovery Services (LUS) as part of a group, having a set of attributes [168]. The ApMon module enables the particular MonALISA server to receive monitoring information. Each discovery service contains a client that listens to these service publications on the MonALISA JINI network, and stores them in an in-memory cache.

The main mechanism used in PingER is the Internet Control Message Protocol (ICMP) echo mechanism, also known as the Ping facility. This allows the sending of a packet of a user selected length to a remote node and to have it echoed back [164]. The server (i.e. the echo responder) runs at a high priority (e.g. in the kernel shell on UNIX) and is therefore more likely to provide a better measure of network performance [127] than a user application. It is very modest in its network bandwidth requirements ( $\sim 100$  bits per second per monitoring-remote-host-pair). Ping is used to measure the response time (round trip time in milliseconds (ms)), the packet loss percentages, the variability of the response time both in the short term (time scale of seconds) and longer, and the lack of reachability i.e. no response after a succession of pings [12].

Once we have collected the monitoring information, we can use it to calculate the values given in the algorithm described in chapter 4 and to populate the cost matrix. Two monitoring tools (PingER and MonALISA) were used because each, by themselves, do not provide the coverage of requirements needed in the scheduling process. MonALISA is an excellent tool for real-time discovery and propagation of service information but it does not give very detailed monitoring information such as that provided by the PingER tool. PingER is a very

useful tool for gauging the network performance behaviour and anomalies detection but, MonALISA is better suited for the decentralized services access which is not available in PingER. MonALISA is also better suited to the discovery service discussed in section 6.4 and the Data Location Service to be outlined in chapter 7.

#### **6.4 Network and Information Discovery**

Discovery Systems (DS) can be considered as an entry point for the Grid and therefore are vital to Grid scheduling decisions. The Discovery System enables the users to find the services and resources for scheduling and executing the jobs. This is the first step in the lifecycle in the execution of a job and therefore is an entry point for the Grid. An efficient DS in essence increases the performance, reliability and decision making capability of the scheduling system, particularly if the complexity is large, as is the case with Grid systems. There can be many cases when information on a service is available within the discovery service but the service itself is either up or down and its availability is not published to other sites so they still have the old information regarding the status of this service. This dynamic behaviour of the services needs to be updated to other sites as quickly as possible. Otherwise these ‘stale’ entries can lead to a degradation of discovery performance and reliability. Thus the registration of services needs to be subject to a lifetime after which the registration expires. Service providers should thus periodically republish information to avoid deletion of previous published information. With the rapid increase in the scale of distributed applications, existing solutions for discovery systems are not particularly effective, for example, when handling the service lifetimes and providing up-to-date status information. They are poor at enabling dynamic service access and they also impose restrictions on interfaces (plug-ins) to widely available information repositories. We also need to provide support for multiple plug-ins to different information repositories, depending upon the requirements of the Grid users. Providing a plug-in API supports flexibility in incorporating new components or in improving existing information services without affecting the overall functionality of the system.

The main reason for opting for a multi plug-in approach is that there are multiple applications which exhibit the same or very similar (discovery) functionality but are used by different communities. UDDI, ebXML, relational databases and LDAP can all be used as registries for storing service information. Applications like MonALISA, RGMA, MDS4, GridICE etc. can

be used for replication of discovery information among the different discovery nodes. These applications are in use by different Grid communities. For example EGEE is using RGMA while OSG is using MonALISA. The pluggable approach enables us to broaden the usage of the discovery service across multiple communities and also to take advantage of different heterogeneous products, many of which exhibit complementary strengths and weaknesses. Here we present the essential design characteristics and an implementation for a DS capable of overcoming these deficiencies to improve scheduling performance. This DS gathers the network and resource information, performs service discovery and propagates this information to all sites. This DS discovers and performs near real-time (as soon as the service goes down and vice versa) discovery of the services and provides the desired fault tolerance and P2P functionality.

The nature of the Grid resources which are being scheduled and allocated by the Meta-Scheduler varies from highly stable to highly volatile. Services may appear or disappear; new services can become part of the Grid and older services can be withdrawn from the Grid. Moreover, the location of the services cannot be foreseen. Discovery or information services enable resources, users and applications to query for services and to retrieve up-to-date information on demand on their location and interfaces. An information service can also act as a facilitator for the interaction between other Grid components; it is impossible for a service to interact with another without knowledge of its location. A DS optimizes the scheduler in a way that an instance of a scheduler running on one site can interact with schedulers running on other sites to enable selection of the optimal execution site. Moreover, it can help the Data Location Service for choosing the best available copy of the required data.

The main problems with current implementations of the DS which limit their use for the DIANA scheduler are: a lack of P2P discovery, suitable fault tolerance and an absence of a dynamic discovery capability for the renewal of service and multiple interfaces. We have created a DS which contains the features described above. This solution is available as part of the Clarens Web Services framework, a Grid portal to run Grid-based analysis applications and services.

#### **6.4.1 Discovery Service and Scheduling**

The DS allows the service data that is published from any site to be retrieved from any where in the Grid network. This functionality makes it suitable to be used as a global information index for scheduling decisions. The DS can provide the network and resource information for static as well as dynamic scheduling. When a user submits a job, its scheduling is carried out based on a number of parameters. These might involve the site policy, particular hardware and software requirements or other considerations [75]. The DS is used to discover Grid resources and network parameters in order to facilitate the matchmaking process when the jobs need to be scheduled. It is to be noted that a push methodology to schedule the jobs is being followed and consequently jobs are scheduled as soon they are submitted to a scheduler. There is no particular interval or frequency for carrying out the matchmaking and scheduling process. Each VO publishes information concerning the resources available at its disposal and the current status of their use. Each site can publish information regarding the computing power it has into the global index; this involves the number of CPUs, processing power available, the number of jobs being executed and the number of jobs in the queue. When a user submits jobs to a meta-scheduler, it does the matchmaking based on the information received from the information service. The matchmaker contacts the information service to find the values of the parameters which are being stored in the information service. These include the service location and status, network parameters (bandwidth, jitter, packet loss etc), the computing power of a site, the job queue size of the site, the number of jobs running and the availability of the execution libraries on a site. The details of the methods to store and retrieve this information are provided in section 6.6. There are also other components that could be involved when scheduling a job such as maintaining user quota and carrying out accounting.

#### **6.4.2 Architectural and Design Characteristics**

The Design of the Discovery Service is based on the P2P paradigm. Since this discovery service is integrated with the DIANA scheduler which itself works in P2P fashion, it is quite natural and easy to follow this paradigm. By allowing Peers to serve each other in the network, P2P technology overcomes many of the limitations in the traditional client-server paradigm in achieving user and bandwidth scalabilities. According to Qiu and Srikant [162],

the performance of a well-designed P2P system actually improves, rather than deteriorates, with the increase of population size (number of clients). Good design should provide better matchmaking and efficient resource utilization. Similarly, Foster and Iamnitchi describe that P2P technology is promising for large content distribution and the convergence of Grid and Peer to Peer (P2P) computing offers many advantages [163] over the traditional client server approaches.

The architectural components (see Figure 6.2) of the DS can be divided into three main layers: the service interface, the wrapper and the repository. The top layer is the service interface which defines different attributes for describing a service and also methods for retrieving information from the service repository. The bottom layer (the repository) provides different variants for persistent storage of service related data. To provide an interface between different kinds of storage systems and the service interface, a middle layer called the wrapper is provided.

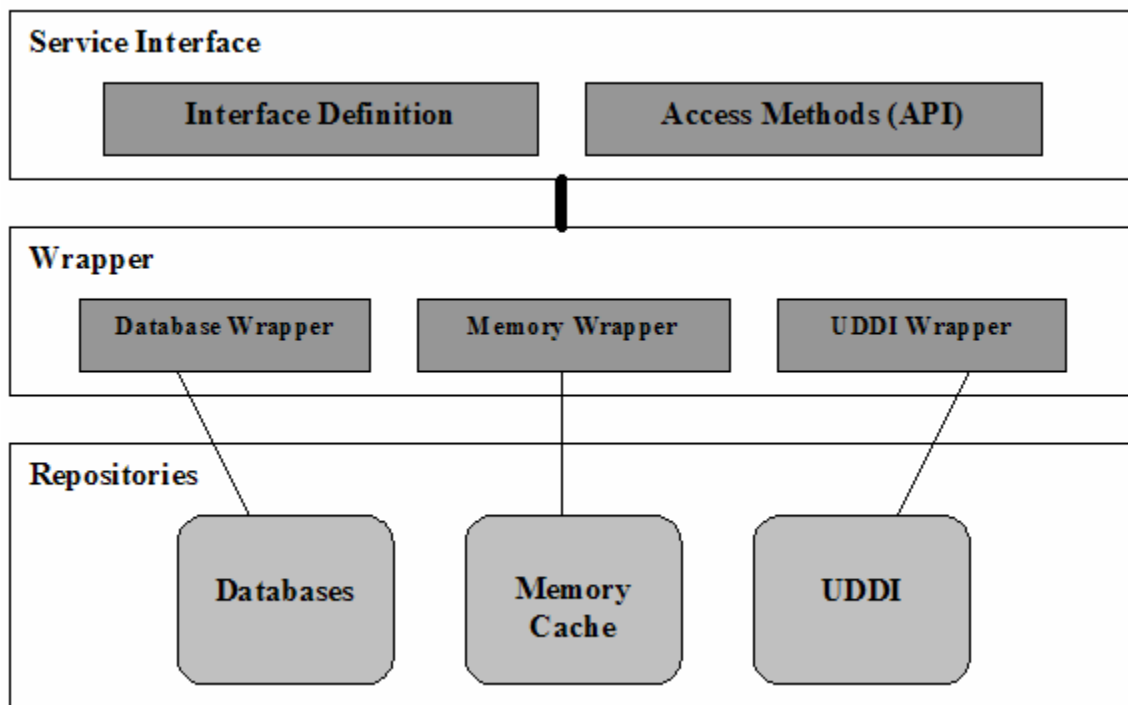


Figure 6.2: Discovery Service Architecture

Repositories provide a mechanism for the persistent storage of service related data. Different sets of repositories provide more or less flexibility for all storage back ends since each of the

repositories is suitable for different types of Grid environments. Relational databases are more suitable in a more stable environment with large numbers of services, where high availability is required. On the other hand if the Grid environment is dynamic with reduced service availability or in cases where a lower response time is desired, then in-memory caches are best suited as storage repositories. One severe limitation of in-memory caches is the need for large amounts of RAM. In-memory caches do not scale well to large Grid environments with large numbers of services. The problem with in-memory cache is overflow when the schedulers register a large number of services or data at a site or across sites. Although service retrieval is faster in this approach it can accommodate only a limited set of services and client requests. The detailed results are given in section 8.3.2 of this thesis. An alternative is to use UDDI (Universal Description Discovery and Integration) - a more standardized approach for service description and discovery. The wrapper layer is responsible for the seamless translation of requests from the interface of the DS to the underlying repository. Another important aspect of the discovery or information service is the ability to co-exist with other instances. This is achieved through the replication of service data over all other nodes of network. The support for global dissemination of service data in a P2P fashion was built by exploiting the capabilities of MonALISA.

MonALISA is a scalable monitoring system and is being used in the DS to monitor the Grid network and services. The scalability of the system arises from the fact that it uses multithreaded station servers for hosting dynamic services. These station servers are interconnected with each other in a P2P fashion and are capable of discovering and tracking any change (transition) in the state of other station servers. The information from the MonALISA can be retrieved using either the “push” or “pull” models. PingER collects the monitoring information from each site and this information is propagated to other sites in the Grid by MonALISA. In the DS, support for multiple plug-ins is provided dependent on the requirements of the Grid users. Providing a plug-in API supports flexibility in incorporating new components or in improving existing ones without affecting the overall functionality of the scheduling system. The main reason for opting for a multi plug-in approach is that there are multiple applications which exhibit the same or very similar (discovery) functionality but which are used by different communities. Another important design consideration is the service information lifetime. There can be many cases when information on a service is available within the discovery service but the service itself is unavailable. These “stale”

entries can lead to a degradation in discovery performance and reliability. Thus the registration of services needs to be subject to some lifetime, after which the registration expires. Service providers should periodically republish information to avoid deletion of previous published information.

This step can significantly improve scheduling performance and reliability. Each independently running instance of an information service should keep aggregated data about all other instances. The benefit of this approach is that information can be retrieved by contacting any of the nodes, without bothering about the location from where the information regarding the service was published. It also allows fast retrieval of service data. Although this offers certain disadvantages in terms of loads on individual nodes the performance gains in term of access can overcome this cost.

## **6.5 Peer-to-Peer (P2P) Topology-Aware Discovery**

In the DIANA scheduling model, each scheduling Peer gives preference to those Peers for communication and information dissemination which have a higher network throughput. A single Peer cannot handle or store the information about each site and node in the centralized Grid system. Therefore, we describe a P2P topology-aware discovery service in which the scheduling Peers will be aware of all the other Peers based on the available bandwidth from each others. This network and topology-aware resource discovery will help the scheduler to make the network aware scheduling decisions and will enable it to peruse the network infrastructure more optimally. In a Grid environment, sites have variable network connectivity and the peers/sites that are closest to each other are the ones which have a low round trip time (RTT) between them. Low RTT brings high bandwidth and hence a faster communication. The proposed topology for the meta-scheduler should lead to a fast and reliable communication between the Peers. Peers will self-organize themselves into sub-Grids and in each sub-Grid, all Peers are nearest neighbours to each other. All subGrids and their Peers will be members of the RootGrid.

To demonstrate our approach, consider that we have six Peers and we need to find the closest Peer, with respect to our site, among these six Peers (numbered 1-6). In order to find the nearest Peer, we have two options: either to calculate only the RTT from these Peers or to calculate available bandwidth to these Peers. In our case, we need more than just the RTT as

the criterion to find the nearest node. The other performance parameters that we use include Packet Loss and Available bandwidth. As described in [126], we can estimate the upper bound of available bandwidth as a function of RTT, Packet Loss and MSS:

$$BW < (MSS/RTT) * (1/\sqrt{\text{Packet Loss}})$$

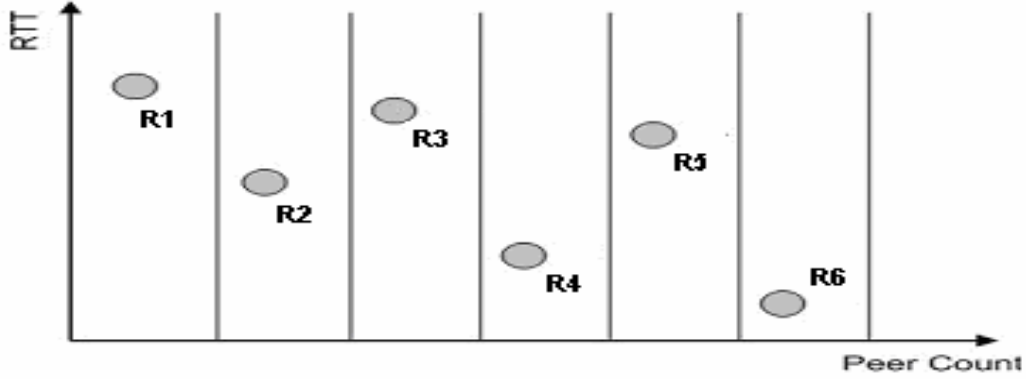


Figure 6.3: Round Trip Time from source 1-6 to destinations

As we see in Figure 6.3,  $R6 < R4 < R2 < R5 < R3 < R1$ ; this shows that Peer 6 is the nearest node and then Peer 4 and so on. But in Figure 6.4, we see that  $B4 > B6 > B1 > B5 > B2 > B3$ . This means that Peer 4 has a better network performance as compared to other Peers from the job submission site. The reason behind these results is that bandwidth does not depend only on RTT, but it also depends upon Packet Loss. In order to define the nearest Peer, let us say we have  $N$  destination Peers and we have to find the nearest Peer among them.  $B1, B2, B3, \dots, B_N$  is the estimated available bandwidth from source to  $N$  destinations. Using this notation, we define the nearest Peer as:

$$NP = \{ B_k \mid \max (B_i) , 0 < i < N \}$$

In this way, the Peer with maximum available bandwidth from source is our NP. The bandwidth is measured between the sites (from source to all destinations or from the site of the job execution to the places where the required data exists); the link from the source to the destination which has the highest bandwidth is treated as the nearest Peer irrespective of its distance from the execution site. The equation above describes this effect and selects the maximum bandwidth amongst the available sites. However if a case arises where there is no major difference in the bandwidth measurement between the sites but the RTT difference is

significant then a link will be selected which has the minimum RTT and the highest (or equal in this case) bandwidth.

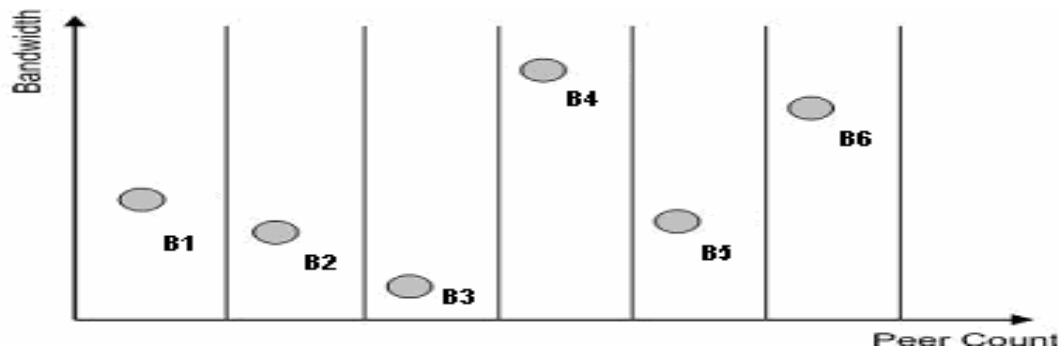


Figure 6.4: Available Bandwidth from source to 1-6 destinations

This P2P approach works as follows: The nodes are divided into SubGrids, each SubGrid having its own "RootGrid" and (on average) each site has one RootGrid and may have one or more SubGrids. A SubGrid is the collection of resources which are geographically proximate and under one administrative control. In most cases each site will be under one SubGrid but more than one site can also constitute a SubGrid if the sites have fewer machines. A SubGrid consists of machines that have good network connections between them. Each SubGrid is represented by a RootGrid, which is the most available machine within the vicinity of the SubGrid. The meta-scheduler works at the RootGrid level in this approach and therefore we use the RootGrid, and meta-scheduler interchangeably to describe this approach. The RootGrid to RootGrid communication is in essence P2P communication between the meta-schedulers. Each RootGrid maintains a table of entries about the status of the nodes which is updated when a node joins or leaves the system and in almost real time. Local schedulers work at the SubGrid level. When a user submits a job, the meta-scheduler at the RootGrid communicates within the SubGrid to find suitable resources. If the required resources are not available within the SubGrid, it contacts the RootGrids of other subGrids in the VO which have suitable resources. Therefore a single node within a SubGrid communicates only with the Meta-scheduler, which itself communicates with the meta-schedulers at other RootGrids.

Therefore, this approach is not just all-to-all communication. Had it been the converse where a machine communicates with all peers in the Grid (all-to-all communication), that would have been prohibitively expensive and the solution would not have been scalable. A

RootGrid contains all information about the nodes in its SubGrid. In case a RootGrid crashes, a standby node in the SubGrid can take over as a RootGrid. The RootGrid replicates its information almost in real time to this standby node to avoid information loss. The RootGrid should always be the machine with the largest availability within that SubGrid and will have a unique ID, which will be assigned at the time of its joining the Grid. After joining, a Peer will check for the existence of the RootGrid. If the RootGrid does not exist, it means this is the first Peer joining the system. That Peer will then create the RootGrid and will join it. If the RootGrid exists then the Peer will automatically join that RootGrid and will search for its SubGrids and will join the nearest SubGrid using the criteria stated earlier.

Whenever a site plans to become part of the Grid, a separate SubGrid encompassing the site resources is created and it the nearest RootGrid and may even create a new RootGrid using the criteria stated earlier. If the site is fairly small in terms of resources, this site may also join some existing SubGrid. The size of the SubGrid and RootGrid and other policy decisions have to be taken by a VO administrator and may vary from one Grid deployment to another. This algorithm will setup the topology, as shown in Figure 6.5.

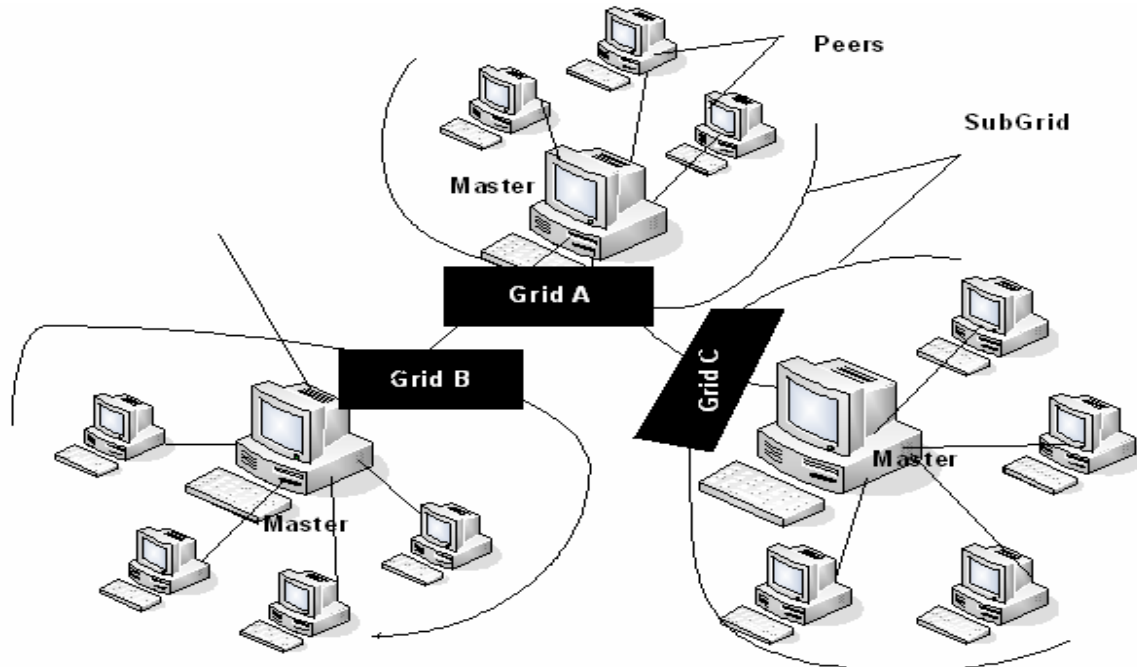


Figure 6.5: Topological Structure

## 6.6 Implementation and API Details

The DS has been implemented as a web service within Clarens/JClarens. The development work has been carried out in two phases. In the first phase the development of a prototype version (or skeleton) of the information service was undertaken. Initially an API is provided with the implementation of the methods shown in Figure 6.6. The service interface (see figure 6.7) provides a way to describe service information. Different methods for accessing and manipulating service data are given in figure 6.6. The choice of parameters was made to have more flexibility in providing service information. The list of parameters is given in figure 6.7 with a list of methods in figure 6.6.

To provide an interface between different kinds of storage systems and the service interface, we provided a middle layer called the wrapper which can interact with any of the existing repositories including UDDI. There are differences in the service APIs but this wrapper can replicate the information between UDDI and our discovery service. This also enables us to remain independent of the specific implementations of the repositories, e.g. the UDDI registry. The wrapper has similar functionality to a UDDI registry, but provides a more lightweight API with xmlrpc accessibility and a much more dynamic and distributed architecture. Service publications can be performed at any instance of the discovery service, and the publication will be automatically broadcast to all other instances of the discovery service. This allows clients to locate services from any discovery service instance, regardless of where the initial service publication took place.

Interface	Description	API Details
Method		
register	Publishes service information with the discovery service.	register(ServiceDescriptor[])
deregister	Removes the service information from the repository	deregister(ServiceDescriptor[])
find	Allow users to query the information service for particular service or services	find(encoding, uri, provider_dn, vo, name)
find_server	Same as find but only provides the	find_server (encoding, uri,

information about existing servers.          provider\_dn, vo)

Figure 6.6: Service access methods

Name	Description
Endpoint list {uri, encoding (soap, xmlrpc)}	Information on how to access the service including protocol and URL
Name	Name of the service
Admin Email	Email of the administrator who is managing the server
VO	The virtual organization of which this server is a part.
Site	Name of the site
WSDL url	URL for retrieving WSDL url
Provider_dn	Distinguished name of the provider of this service
Item {key , value}	Arbitrary key-value pair for describing the properties of the services

Figure 6.7: Service description

The data on each service was stored at a centralized location. A discovery module was provided as a core component within the Clarens framework. Each instance of a Clarens server registered its services with the locally available information service. Replication was not possible between servers and consequently no communication was possible among different Clarens servers.

The support for the replication of the service data was built and communication among individual instances was provided in phase 2, as shown in Figure 6.8. A new monitoring module was developed within the MonALISA which handles publication requests from individual instances of Clarens. Upon its start-up each Clarens server registered its services with the local repository and also pushed them to the MonALISA server. The service

information is sent to any of the known station servers using the ApMon client library – a library that uses simple XDR UDP packets. A JINI-based client is embedded within each Clarens server. This client registers with the MonALISA JINI network to listen for any kind of information related to the DS. The values are pushed to the Clients so we do not have to periodically check for new or modified values. The information received from MonALISA is also stored in the local repository. In this way service information from one discovery instance is made available to others.

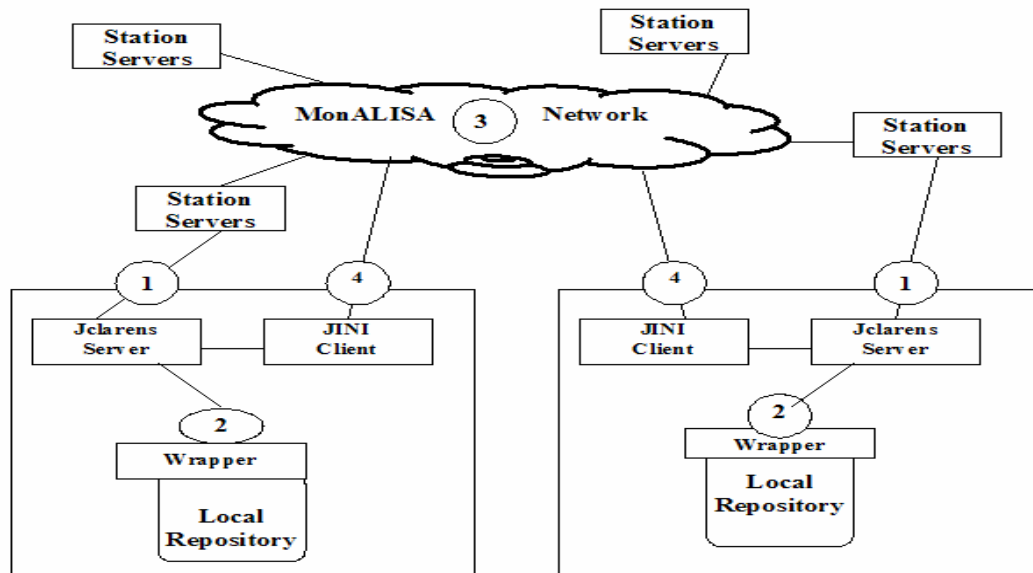


Figure 6.8: Service replication and distributed discovery

One other important development is to add a service lifetime management mechanism. Keeping information for unavailable services not only decreases the efficiency of the network, but it may also hinder the overall working of the scheduler. Thus lifetime management can be considered a critical component of the DS. In the implementation of the DS, the publishing of service information is subject to a finite lifetime. Repositories are periodically purged to remove expired entries from the system. The service provider is responsible for renewing the service lifetime; the service expiry is managed by the provider of the service, so it can be easily adjusted based on the requirements and nature of the Grid applications.

## 6.7 Conclusion

Chapter 6 presented the effect of the network parameters and network characteristics on the Meta-Scheduling process. No data-intensive system can be successful so long as it does not coordinate with the underlying network. Network aware scheduling has been discussed and its influence on scheduling decisions has been analyzed. This chapter also described the design and implementation of a DS. We discussed some unique features of the system - the most important of which include flexibility, avoiding stale service data by imposing a lifetime for each service entry and providing an updated view of the dynamically varying Grid system. The ability to attach arbitrary sets of key-value pairs with service information makes it possible to easily extend/customize the service description interface. The DS is a core component of the DIANA P2P scheduling system.

In chapter-7 we will discuss Data Aware Scheduling. The role of the Data Location Service and its key attributes are explained. Chapter 7 will investigate how catalogues perform in the DIANA Scheduler and what are the key features of DIANA which make it unique for Data Intensive Scheduling. Chapter 7 also sheds light on the replication and co-allocation issues for Data Intensive Scheduling and describes the ways through which this is managed in the DIANA Scheduler.

## **Chapter 7**

### **Data Management and Scheduling Optimization**

Chapter 6 discussed the network characteristics and network managed services which can influence the data intensive scheduling process. Network characteristics can significantly influence the data intensive scheduling process and therefore, it should be treated as prime resource while making the scheduling decisions. The network issues and their association with the meta-scheduling was established and the role of a Grid discovery and information system was illustrated to dynamically manage the services and network information. Chapter 6 also presented the architecture and salient features of the Discovery Services. Finally the role of a Discovery Service as a P2P enabler for DIANA scheduling was described.

In chapter 7 the data related aspect of DIANA scheduling are described. The Data Location Service (DLS) and its key features are explained. The role of catalogues in the DIANA Scheduling and the key features of the DIANA which make it unique for data intensive scheduling are highlighted. Chapter 7 also sheds light on the replication and co-allocation issues for data intensive scheduling and describes the ways through which this is managed in the DIANA Scheduling System.

#### **7.1 Introduction**

Grid deployments are facing problems due to the limitations in its data management capabilities. This is due to the fact that the word "Grid" very often refers to a compute Grid with no explicit reference to data semantics. As such, data management is typically an afterthought in many of these deployments and data operations need to get a priority due to the increasing number of the data intensive scientific and business applications of the Grid. A simplified architecture of the Grid system is illustrated in Figure 7.1 [154] to describe how compute and data resources interact. In this example tasks from Grid clients are received by a Grid scheduler, which then routes and manages these client requests to a set of compute nodes. The compute nodes in turn access the necessary information from backend data sources. The challenge with most Grids is that the computations they support are fairly data intensive and have to be executed with a high degree of parallelism, such as some of the Monte Carlo simulations for physics analysis [153]. Often there is also a need to share intermediate results amongst a set of interdependent Grid tasks. Data sharing and access can

also be very cumbersome since due to better computing power, it may be preferable for the jobs to run at a site different to the data location. As a result, most Grid deployments suffer from data latency, scalability, consistency and quality of service issues. While these issues are seldom noticeable in small-scale pilot deployments (of the order of 10-20 nodes), the moment an organization plans a Grid expansion both in functionality and scale (to, say, hundreds of nodes), such data management issues become more pressing. To combat these issues, we need to recognize the need for innovative data services which can serve the needs for data intensive Grid scheduling. Therefore we require services that can deliver this functionality and enable the intelligent, on-demand delivery of actionable information to the meta-scheduler.

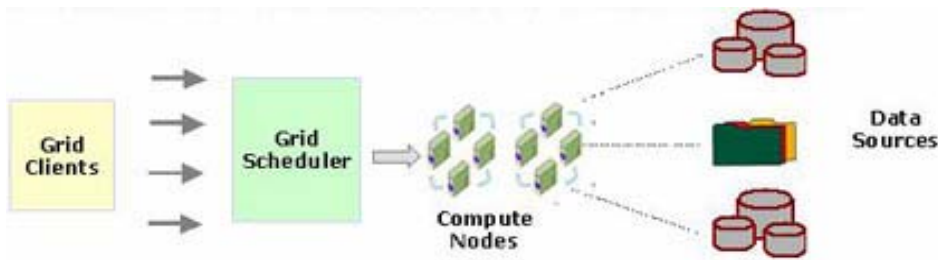


Figure 7.1: Typical Grid architecture [adopted from the 451 Group [159]]

## 7.2 Data Management and Scheduling

Data intensive jobs can sometimes read or require users to store their output data in some secondary storage or tertiary storage at some remote site after the completion of a job. Ideally, end-users should not be concerned where their files are and this should be provided automatically by a service that is transparent to the user. Some applications are required to perform analyses on datasets geographically distributed across many files, and this requires an efficient means to locate and access datasets irrespective of their location and file organization. When building a Grid, the most important aspect is to align (co-allocation or co-scheduling of data and compute resources) the data with the compute cycles. The Scheduler should consider both the data and compute resources for scheduling instead of scheduling from a computing perspective only. Therefore in its design, we need to determine our data requirements and how we will move data around our infrastructure or otherwise access the required data in an efficient manner. Data management is concerned with

collectively maximizing the use of the limited storage space, networking bandwidth, and computing resources and this has a profound effect on scheduling decisions.

For large datasets, it is not practical, and might be infeasible due to storage and transfer limitations, to move the data to the system where the job will actually run. Using data replication or otherwise copying a subset of the entire dataset to the target system might provide a solution. If the target Grid is geographically distributed with limited network connection speeds, one must take into account design considerations around slow or limited data access. For instance, if a data transfer is going to take five hours and the data is required by a job that must run at 14.00 hrs, then one should schedule the data transfer in advance of 9.00hrs so that it is available by the time the job requires it. This is not a resource reservation but rather a scheduling mechanism which takes into account the anticipated transfer times. In resource reservation, a set of resources controlled by the scheduling system are set aside from normal use for the jobs, and made accessible to a subset of users who have been given exclusive rights for this reservation. The resource is not being reserved for a particular user or job in the DIANA meta-scheduler but rather a transfer operation is started in advance to initiate the job execution in time. The users should also be aware of the number and size of any concurrent file transfers to or from any specific resource. The scheduling system will require logic to handle the selection of the appropriate replica that will contain the data that the jobs need, while also fulfilling the performance requirements. Note the replication helps in achieving systems performance. When a scheduler generates a request for a file, large amounts of bandwidth could be consumed in transferring the file from the server to the client. Furthermore the latency involved could be significant, considering the size of the files involved. The replication reduces access latency and bandwidth consumption. Replication can significantly help with scheduling optimization and can improve reliability by creating multiple copies of the same data.

### **7.2.1 Virtualization and Scheduling**

A data Grid connects a collection of geographically distributed computer and storage resources [116] and enables users to share data, compute and other resources. Data virtualization plays an important role in realizing data intensive scheduling. By dynamically managing data across multiple physical nodes, a data Grid converts every compute node in a

Grid into a "data node" as well. These physically disparate data partitions are exposed to a scheduler as a unified logical entity that appears to originate from a single source. The application is not concerned with the actual data location, nor does it adhere to different access protocols, this concept being often referred to as "location transparency" in the Grid environment.

The second important issue is high availability. While improved performance is indeed a major benefit of managing data, it should not come at the price of inconsistent data availability. High availability without perceivable loss in performance is guaranteed through a replication scheme. Any number of data mirrors or replicas can be configured to ensure the required level of data availability. When a primary data node fails, a replica can handle client requests. Therefore, in data aware scheduling, data virtualization and its availability are important considerations and can play a significant role in the scheduling decisions. Figure 7.2 illustrates the storage and handling of data in the data Grid system [128]. The data Grid software components are responsible for wide-area data handling and replication in terms of files. The Grid software components provide a grid-wide file replica catalogue service which maintains the mapping from logical to physical files.

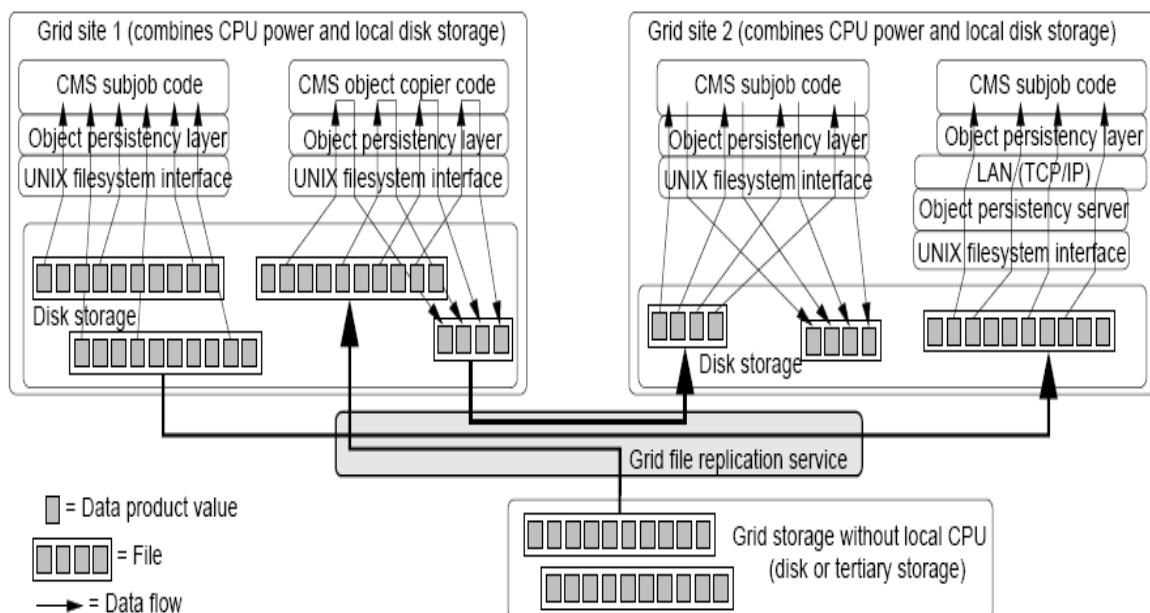


Figure 7.2: Storage and handling of data in the data Grid system

### 7.2.2 Co-Scheduling and the Data Scheduler

The ability of a Data Grid to locate and distribute data in a dynamic fashion facilitates intelligent interactions with other technologies in a Grid. A Grid scheduler can be configured to enable data-aware routing [122]. In other words, by recognizing the data placement strategy of a Data Grid, a Grid scheduler can route certain tasks to nodes that already have the required data available to complete a specific task. Conversely, by understanding the data requirements of a computation being repeatedly executed on a node, the scheduler can intelligently pre-load the data required for that task which often results in significant performance benefits. Based on data access, load concurrency and quality of service requirements, a scheduler can work in concert to locate data on a set of Grid nodes that are equipped with the requisite CPU and memory. This whole process is known as co-scheduling of compute and data operations. An assumption we need to make is that data is usually co-located (and co-scheduled) with the application that needs it. In order to be able to do so, the Grid job scheduler needs to invoke the Data Scheduler services [114] in order to make sure that a given file is available at the chosen site where the job is to be run. This co-scheduling may not be always possible, so it may be necessary to access the data remotely.

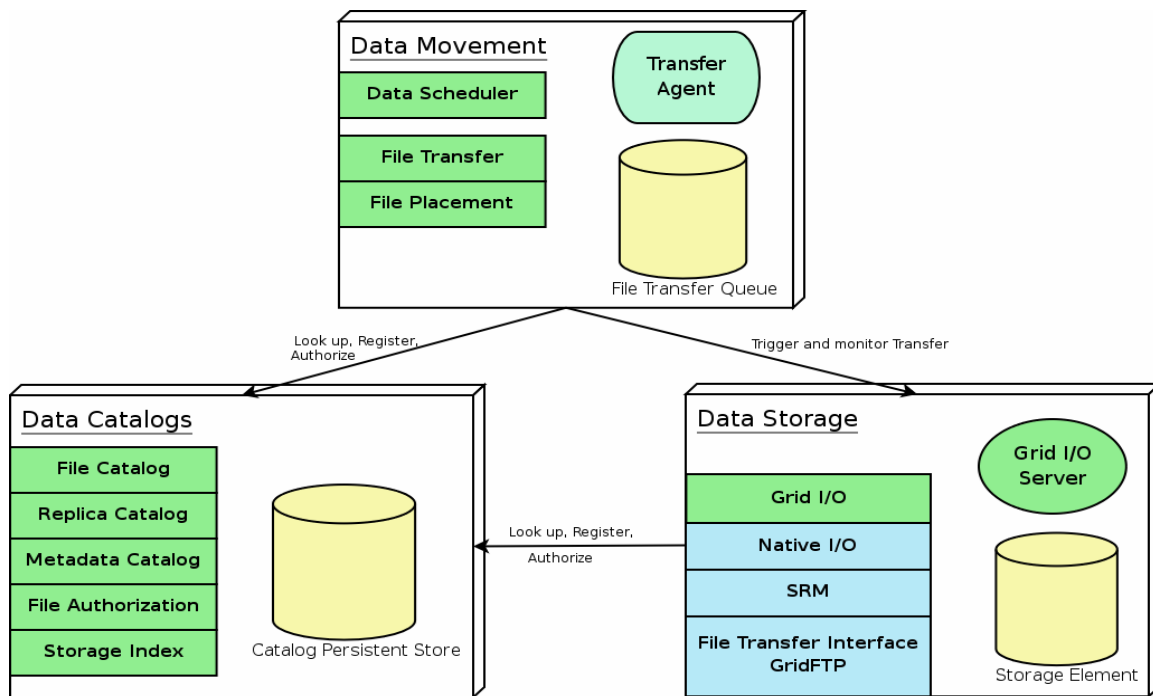


Figure 7.3: Catalogs for data access

### 7.2.3 The Catalogs and the Scheduling Process

In order to achieve the promise of ubiquity, the Grid Resource Management System (GRMS) behaves like a simple operating system to the application. The GRMS performs almost the same functions of resource management at the application level in the Grid that an operating system performs at the kernel level to manage the system resources. Jobs are scheduled, queued and aborted, resources are allocated and released, policies are enforced, authentication and authorization are performed and files are managed in almost the same way at the application level as an operating system does at the kernel level. Therefore GRMS is viewed as an operating system to the application. It should enable applications to access the data independently of the actual location of the CPU on which the job is being executed. Due to the distributed nature of the Grid, the files may be replicated at many Grid sites if they are heavily used. The user application does not need to know which locations these are as long as it is able to run the jobs and read the data as if it were local.

The Grid catalogues make sure that the file names and associated metadata are properly accessible and secured for the end-user application. Using an external catalogue to keep track of the physical location of the file is more flexible than hard-coding the location information in the file itself. This allows the file to be moved or replicated. In the Grid environment, high level applications view only logical files. A logical file does not exist physically; a physical file and all its copies can be viewed as the same logical file. In other words, physical files are representations of the logical file. To operate on the contents of a logical file, a service is needed to map the logical file to one of its physical representations and this physical file can then be opened. The physical location of a file can be represented by its Physical File Name (PFN) while the logical file can be represented by a unique logical identifier. It is not mandatory for each Grid site to provide the storage functionality described in figure 7.3 but each storage site in the Grid should provide this functionality to provide replica, metadata and other services for locating and transferring the files for data intensive scheduling. The Grid catalogues are used to manage the Grid file namespaces and the location of the files, to store and retrieve metadata and to keep data authorization information for the scheduler. We can decompose the catalogues (see Figure 7.3 [114]) into catalogue feature sets which are represented by catalogue interfaces. The Replica catalogue exposes operations concerning the replication aspect of the Grid files. The File catalogue allows for operations on the logical file

namespace that it manages. In order to implement a standalone Metadata catalogue, methods are needed to add and remove entries in the Metadata catalogue. The catalogues increase performance and optimise interaction with the Grid scheduler. Figure 7.3 also illustrates the interaction of the catalogues with the indexing, authorization, and data storage and data movement services.

### **7.3 Data Location Service**

Data intensive scheduling takes into account large amounts of data, data replication, and data transfer as illustrated above. In order to optimize the scheduling process in which large amounts of data are involved, we therefore have to take into account data location, size and other related aspects. To address this issue, a Data Location Service (DLS) is created which returns the best replica of a given logical dataset and it uses the Data Location Interface [149] (see section 7.3.2) for this purpose. The DLS provides the means to locate replicas of data in the distributed computing system. The DLS is indexed by datasets logical names (LFN) and it maps datasets to storage elements (SEs) where they are located. Each LFN is linked to a GUID (Grid Unique Identifier) which is unique across the entire Grid. Against each LFN, the DLS locates the GUID which in turn looks up GUID to PFN mappings for a given GUID. The DLS selects the best replica from all the PFNs which are associated with a GUID. The basic content of the catalogue is the one-to-many mapping of the LFN and the PFNs; this mapping is sufficient for object lookup and navigation. To register a file is to insert a LFN and PFN pair in the catalogue while to lookup a file is to resolve a LFN into a PFN or vice versa. Besides, one can extract a subset of files from one concrete catalogue and cross populate them into another catalogue. The DLS provides the names of sites hosting the data and the physical location of constituent files at the sites, or the datasets. Information is entered into the DLS, for sites where the data is located, either by the scheduling system after job execution or by the data transfer system after transfer. Site manager operations may also result in modifications to the DLS, for instance in the case of data loss or deletion. In order to steer jobs to the data, the DLS is contacted to select the sites hosting the needed data. This translates into an explicit requirement to the Grid Meta Scheduler to schedule the jobs at a possible set of sites.

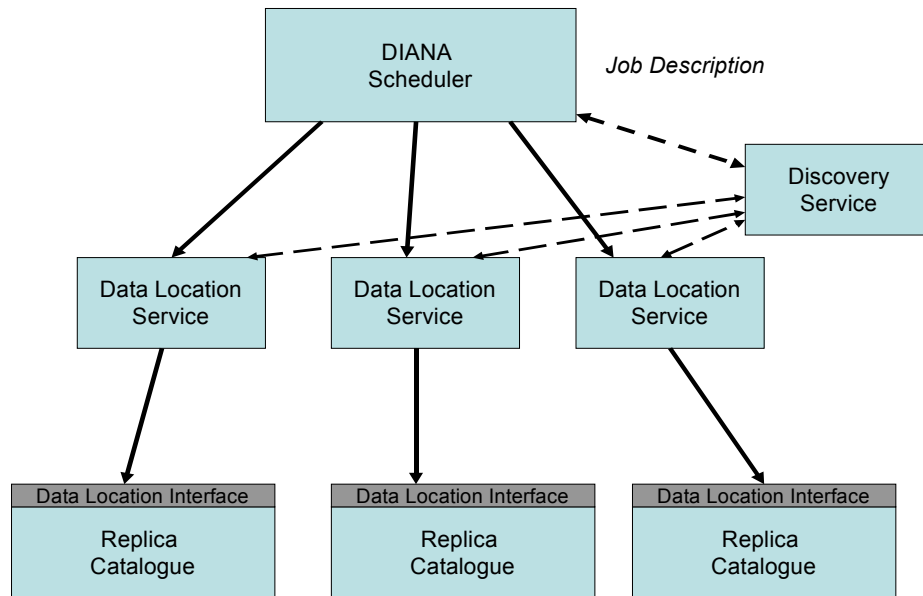


Figure 7.4: Interaction of the DIANA Scheduler with the Data Location Service

Figure 7.4 shows the architecture of the DLS. In this figure, three instances of the service are shown to illustrate that there can be more than one instance of the DLS running at different locations in the Grid. We can obtain the list of these instances through the Discovery Service (see section 4 of chapter 6) which is the point of contact to access and query the DLS. All instances of the DLS returned from the Discovery Service are interacted and queried but physical replica from that location is selected which is likely to have a least data transfer cost. If a client (in our case the DIANA Scheduler) needs to get information about the datasets stored in storage elements and registered in replica catalogues known by the DLS, the client first contacts the Discovery Service which provides a list of all the locations where a DLS is running. Then the client queries the DLS by passing a logical name of the dataset whose physical location is desired. As a result of this query, the DLS returns an optimal physical replica of the dataset to the client with reference to the computing element where the job will be executed.

The DLS outlined here focuses on the selection of the best replica of a selected logical file, taking into account the network and storage access latencies. The DIANA Scheduler selects the best CE and then the best SE, with reference to this CE, is selected on the basis of criteria as discussed in chapter 4. The DLS is a light-weight web service with few SOAP and other

custom libraries to copy for installation and running on top of the existing Grid catalogues and monitoring infrastructure. It gathers information from the network monitoring service and performs access optimization calculations based on this information. The DLS provides optimal replica information on the basis of both faster access and better network performance characteristics. The data location process allows an application to choose a replica, from among those in various replica catalogues, based on its performance and its data access features. Once a logical dataset is requested by the Scheduler, the DLS uses a replica catalogue to identify all replica locations containing physical dataset instances of this logical dataset, from which it should choose an optimal instance for retrieval. The DLS is fault tolerant, so that when one instance goes offline, a scheduler or any other service can still be able to work by using another instance of the service. All the instances of the DLS are registered in the Discovery Service and it is simply a matter of making just one SOAP call to the Discovery Service to find another instance of the DLS. All the instances of various services including the DLS are updated after regular intervals, therefore it is guaranteed that there are no stale entries in the Information and Discovery Service. As discussed in chapter 5, the exact interval to collect the updated services and network information can be changed by the site administrator through changing the value in the configuration file but a 15 minutes interval was used to update the values. This behaviour is provided by MonALISA and the information service as discussed in chapter 6.

The DLS is a decentralized service which takes into account the selection process on the basis of both scheduler and dataset locations and associated network parameters while using the discovery service to locate the available replica catalogue services. Each of these catalogues is queried by the DLS to find all locations where the requested dataset is available. The service returns a list of dataset locations to the caller. The result of a call to this service is sorted either by the reliability of the datasets (which is provided by the network cost and network features such as bandwidth, packet loss etc) or by the "closeness" determined by some network ping time or other network measurements. The DLS also evaluates the network costs for accessing a replica. For this it uses information such as estimates of dataset transfer times based on network monitoring statistics. The DLS selects the "best" physical dataset based on the given network, size and location parameters.

### 7.3.1 Architectural and Implementation Details

The complete system (that is the DIANA Scheduler with the DLS) is implemented in Java although components employed use C, Python and Perl to incorporate the libraries which have been developed under different Grid middleware projects like Globus and gLite. We use SOAP as well as XML-RPC as the communication protocols to send requests to the services. JINI based MonALISA is the core provider of the P2P behaviour in the Discovery Service and it inherits parts of the functionality from JINI. The use of MonALISA in the Discovery Service was discussed in section 6.6 of chapter 6. Furthermore, the detailed description of the decentralized functionality is also provided in the same chapter. We have employed PingER to get the monitoring information since it provides detailed historical information about the status of the networks. For the time being, PingER does not provide the web service interface, so we store the network monitoring information in a MySQL [108] based relational database and simply access this database for using this monitoring information. These monitoring values are published to the MonALISA using Apmon API as was discussed in the chapter 6. A module incorporates the networking information in the scheduling algorithm. Moreover, we created a multi-dimensional array to store the cost values for various nodes and populated the cost matrix. We used the search routine discussed in chapter 4 to get the best cost for job execution. The Matchmaker, meta-scheduling components, Clarens based Discovery and Information Service and itself JClarens (A Java variant of Clarens Framework) for services deployment, Data Location Service, Job Monitoring Service, Job Steering Service and Estimator Service are created for this thesis whereas the rest of the components and utilities have been used from other projects. SOAP and XML-RPC are used to provide the request-response mechanism between all these services. The same protocols are used to provide access to the contents in the MonALISA repository.

The DIANA scheduler makes use of a peer-to-peer network to find and schedule the resources on the Grid. The current implementation, as shown in Figure 7.5, makes use of three software components for resource discovery: Clarens as a service deployment environment, a Discovery Service which stores and publishes the services and network parameters,, and a peer-to-peer network provided by JINI based MonALISA network to propagate the information in the Discovery Service across the entire Grid.. In DIANA, Clarens provides a common resource registration service called the Discovery Service (as

previously described). Resources such as Data Location Service and other web services that are offered by Clarens are advertised directly to the Discovery Service. Resources offered by non-Clarens software packages can also be registered with a local or remote Clarens Discovery Service. The Discovery Service forwards the resource description to a MonALISA Station Server, where it is propagated to all other Discovery Services by means of MonALISA's underlying JINI network.

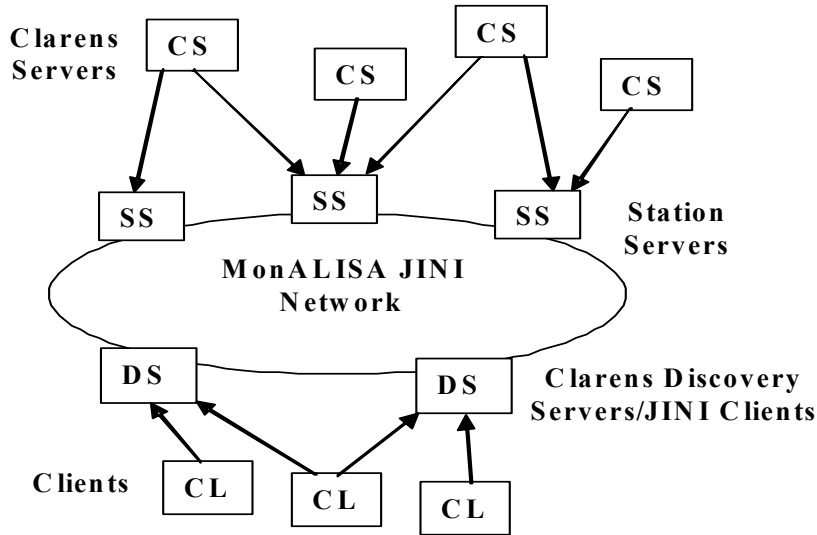


Figure 7.5: Clarens Servers (CS, DS), MonALISA Station Servers (SS), and clients (CL) as part of a P2P discovery system.

Each Discovery Service receives all resource publications from all other Discovery Services, providing a global view of all available resources in each Discovery Service. The DIANA Scheduler can then contact any instance of the Discovery Service to find out about available resources on the entire Grid. If any instance of the Discovery Service becomes unavailable due to hardware or network faults, then any other instance of the Discovery Service can be used in its place. In fact, the Discovery Service can also be used to locate other instances of the Discovery Service. This helps to work around the bootstrapping problem as a client application only needs to know the location of a single available Discovery Service. Once that service has been contacted, a list of other Discovery Services can be cached locally and used if the original Discovery Service ever becomes unavailable.

In order to fully include the Data Location Interface (DLI) (see section 7.3.2) into the scheduler, changes were required on the client as well as on the server side. The main change is the integration of the DLI client into the Matchmaker. DLI calls follow generally accepted web service standards. For each virtual organisation (VO) one has to specify if either the Replica Location Service (RLS) or the DLI is called by default. Consequently, when a data intensive job request arrives, the Meta-Scheduler checks the InputData type and the server side configuration in order to distinguish between RLS and DLI calls.

### **7.3.2 Data Location Interface (DLI)**

The Data Location Service uses the DLI to access the replicas from the various catalogues [149]. The DLI is also a Clarens Service just like the Data Location Service. The DLI does not directly interact with the Discovery Service. Rather the Data Location Service coordinates between the DLI and the Discovery Service. For example, the location and contents of the DLI are not published to the Discovery Service. Instead, the Data Location Service queries the catalogues through the DLI and publishes the dataset's information to the Discovery Service. The underlying protocol of communication between the DLI and Data Location Service is SOAP and the Data Location Service also queries the Discovery Service in the same protocol. A unique feature of the DLI is that it locates datasets rather than individual files. A dataset is considered to be an atomic unit of data that is defined within a VO. Furthermore, a dataset itself can consist of several physical files but the end-user (for example a physicist) normally only knows the dataset concept. The DLI (see figure 7.6) takes a Logical DataSet (LDS) name as an input and returns the physical locations of the datasets. The LDS is defined to refer to an entity of data or a so-called "file collection". One LDS can have several physical replicas. For simplicity we assume that all physical files that belong to a dataset are stored at the same storage element and are accessible via the same protocol.

When a scheduler wants to obtain all physical locations for a given LDS, a SOAP request is sent to the Data Location Interface via a Data Location service. The Data Location service issues the SQL query to retrieve the possible locations where the dataset is available. The results are then mapped internally by the service to have the corresponding Storage Element hostnames. Once this is done, a SOAP answer conforming to the Data Location Interface is created using the ZSI python module, and is then returned to the scheduler.

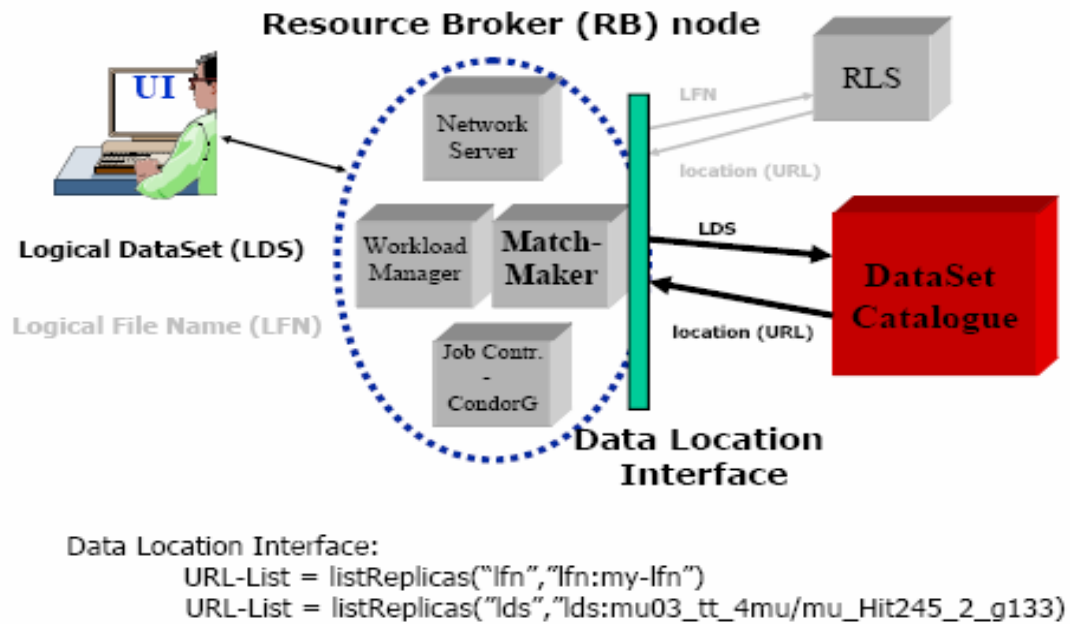


Figure 7.6: Data Location Interface (adopted from [149])

Conceptually, logical files names (LFNs) and LDS can be treated in the same way by the scheduler. For both data types, LFN and LDS, it is important to know where the physical replicas are located. For LFNs, the replica location is returned by the Data Location Service. For logical datasets, an equivalent service needs to implement such a query method and this is provided by the DLI that acts as the main interface between the DIANA Scheduler and a catalogue for a specific data type. Therefore the DLI is designed in Web Service Description language (WSDL) and consequently, the Scheduler can interact with any catalogue (providing data locations) that exposes a web service interface.

## 7.4 Conclusions

Chapter 7 described the data related aspects of DIANA Scheduling. It was shown that the scheduling efficiency can be improved by selecting the best replica of a dataset having the minimum access and transfer costs. The DLS enables the selection of this ‘best’ replica of a logical file by taking into account the network and storage access latencies. Moreover, the relation between the network characteristics as explained in chapter 6 and data intensive scheduling was established and it was shown that the data intensive scheduling process can be optimized by taking the replica locations and sorting these replicas based on the network

efficiency. The dataset replica having the least access cost with reference to the selected computing element is selected by the Meta-Scheduler. Furthermore, the DLS exploits the Discovery Service to locate the dataset and file catalogues.

In Chapter 8 results of tests are presented. Through graphical and analytical details it will be demonstrated that DIANA Scheduling significantly optimizes the job queue and execution times. It will also be explained that bulk job scheduling and execution operations are improved by DIANA Scheduling. It will be illustrated through these results that P2P is better suited to data intensive, bulk job scheduling. Moreover the impact of the network and data location will be demonstrated for replica selection and consequently on the meta-scheduling optimization.

## Chapter 8

### Results and Discussion

This chapter presents results of practical tests of the thesis investigation. Section 8.1 describes the sequence and approach followed for the results. Section 8.2 presents the results of the DIANA deployment on GILDA (the Grid INFN Laboratory for Dissemination Activities testbed). Section 8.3 describes the results regarding P2P scheduling on a custom-built testbed and it also demonstrates job steering and information and migration processes for the optimization of scheduling. Section 8.4 discusses bulk scheduling and other simulation results which have been taken from the MONARC [63] and SimGrid [45] based simulation frameworks.

#### 8.1 Introduction

Having presented the theory, mathematical background and implementation details of various services for DIANA scheduling, the results of practical investigations are now detailed. In this chapter, we present a list of the results which were achieved through experimentation and simulations as follows:

1. Firstly we present the DIANA Scheduling results demonstrated through the use of the GILDA testbed created by the EGEE project. In section 8.2, we present the testbed set-up and then we discuss our findings about job execution times, queue times and replica selection and transfer times.
2. Secondly we show how we created our own experimental testbed to obtain results in which the P2P capability of the DIANA Meta-scheduling is demonstrated. In section 8.3 we also demonstrate the results of the DS that shows how effectively it registers, discovers and replicates the peer's information. Further results related to priority control, queuing and job migration when a scheduler is overloaded at a site, followed by monitoring and estimation results are presented in this section.
3. Thirdly the limitations posed by the testbeds in steps 1 and 2 above are described and how they were overcome by creating a simulation environment is detailed. In particular the bulk scheduling and scalability process is demonstrated with the aid of simulation results. This is

presented through simulation tests since it is not practically feasible to submit thousands of test jobs on the Grid testbed since these tests can take days to produce basic bulk scheduling and scalability measurements. Consequently in section 8.4, the simulation results for bulk scheduling, export and import of jobs to and from the remote sites and scalability tests are also presented.

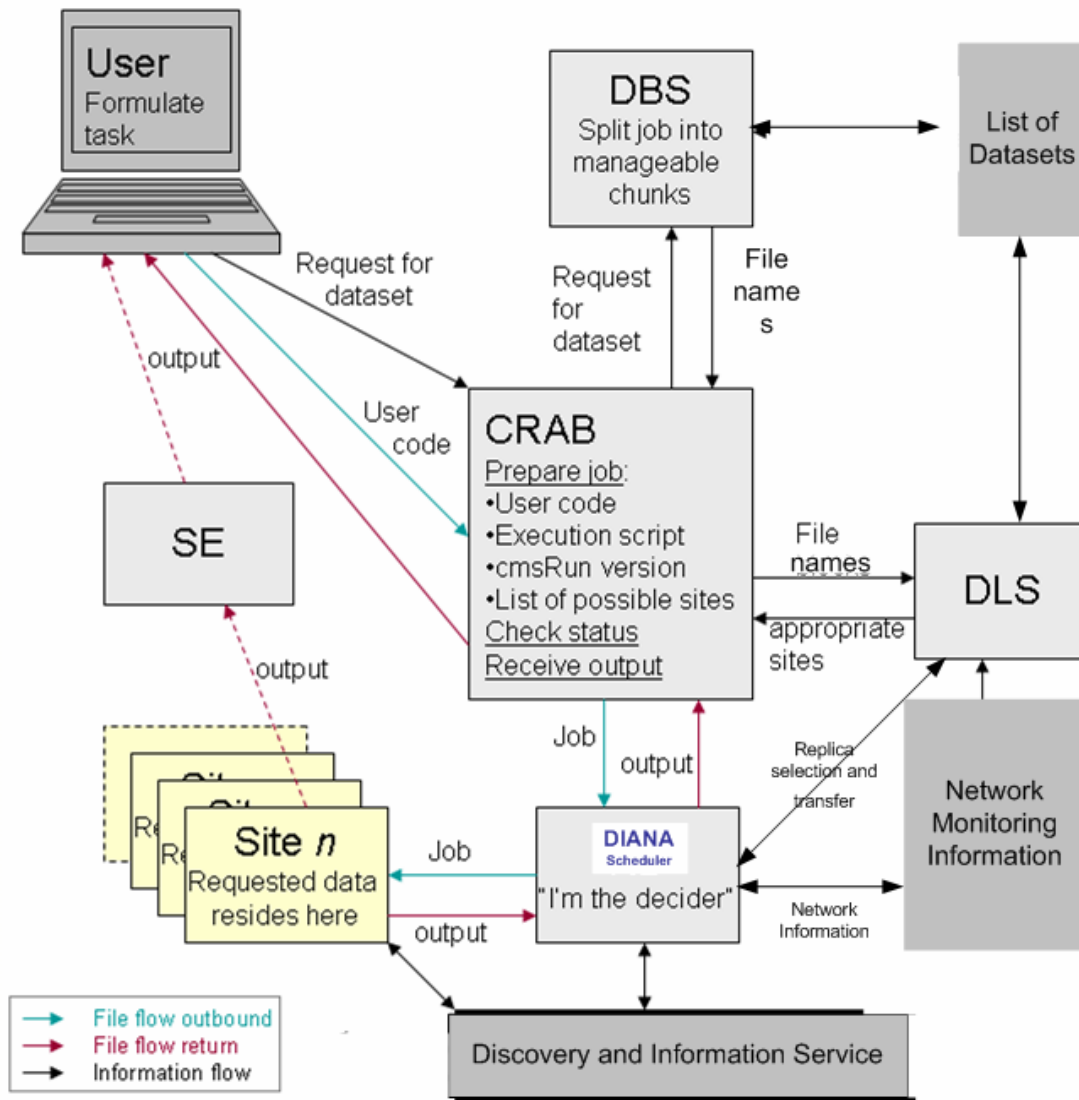


Figure 8.1-a: A description of the experimental environment

## 8.2 Experimental Results through the GILDA Testbed

CRAB [165][168] has been used as a job creation and submission tool in the experiments; CRAB is short for CMS Remote Analysis Builder. CRAB enables CMS users to easily perform analysis jobs on every data or MC (Monte Carlo) set officially published by CMS. CRAB hides the usage of the Grid middleware from the user who is not required to have any special expertise about the Grid. The users use CRAB to run jobs on officially published datasets. CRAB takes care of the submission, the job splitting and the output retrieval. Users use CRAB to create jobs in order to process a defined number of events of the requested dataset, to submit the jobs via the Grid to farms where the requested dataset is available or can be made available and to check the status of submitted jobs and to retrieve the job output. Figure 8.1-a shows the experimental environment which describes the flow of user code, physics data, and job- and resource-related information throughout the course of an analysis job.

The scheduling system will create job configurations for every job which is to be submitted. At submission time, the submission tool will have information about the data location and will pass this information to the DIANA Scheduler which in turn can decide where to submit the job according to some resource availability metrics. The Scheduler will submit the jobs to the Grid as a "job cluster" if necessary, for performance or control reasons, and will interact with the job monitoring to allow the tracking of the submitted job(s). The DIANA scheduler is responsible for scheduling the jobs to run on specific Compute Elements (CE) and dispatching them to the CEs. Each job run-time takes place on a Worker Node (WN) of a specific Computing Element (CE)/local resource management. The jobs arrive on the WN with an application configuration which is still site-independent. The CE/WN is expected to be configured such that the job can determine the locations of necessary site-local services (local file replica catalogue, CMS software installation on the CE, access to CMS conditions, etc.). Once the job completes, it must store its output some place. For very small outputs, the outputs may just be returned to the submitter as part of the output sandbox. For larger outputs, the user will have choices. Either the output can be stored on the local Storage Element (SE) (for subsequent retrieval by the user) or it can be handed off to an agent of the data transfer system for transfer to some other SE. In any case, handling of the output data will be asynchronous with respect to the job completion. The job's only obligation is to either

successfully store the outputs to the local SE or pass them to the data transfer agent. While job processing is in progress, the user can monitor the progress of the jobs constituting his task by using the job monitoring system.

As individual jobs finish (or after the entire set of jobs in the task has finished) the user will find the resulting output data coalesced to the destination specified during the "job completion" step. If the user wishes to publish this data, the relevant provenance information must be extracted from the job monitoring/bookkeeping system, etc. The location of the resulting file blocks can then be published in the DLS. Publication is not (yet) supported by CRAB. These pieces thus constitute a basic workflow using the CMS and Grid systems and services. The CMS tools in association with the DIANA Scheduler are responsible for orchestrating the interactions with all necessary systems and services to accomplish each specified task.



















































Site ▼		Computing Resources										Storage Resources		
		GK#	Q#	RunJob	WaitJob	SlotLoad	MH#	Power	WN#	CPU#	CPUload	Available	Total	%
CECUM-ME		1	3	0	0		2	12K	1	2		194.3 GB	203.3 GB	
CENAPAD-UNICAMP		1	3	0	0		3	3K	1	1		59.7 GB	65.9 GB	
CNR-ROMA		1	3	0	0		6	4K	3	3		12.4 GB	17.2 GB	
DIST-GENOVA		1	3	0	2		5	38K	3	8		360.8 GB	370.5 GB	
IHEP-BEIJING		1	3	0	0		-	-	-	-	-	51.2 GB	55 GB	
IISAS-GILDA		1	4	0	3		8	24K	5	5		27.2 GB	28.9 GB	
INAF-CATANIA		1	3	0	0		3	3K	1	1		103.1 GB	106 GB	
INFN-CATANIA		1	3	0	0		13	201K	9	36		3 TB	3.9 TB	
INFN-CNAF		1	4	0	0		4	6K	1	2		206.9 GB	231.4 GB	
INFN-PADOVA		1	3	3	0		8	42K	6	12		330.9 GB	332.9 GB	
ING-MESSINA		1	3	0	0		6	36K	4	6		503.6 GB	521.1 GB	
IUCC-LCG2		1	3	0	0		7	16K	5	10		857.1 GB	870.1 GB	
TRIGRID-UNIPA		-	-	-	-	-	2	-	-	-	-	3 GB	4.6 GB	
prague_cesnet_gilda	-	1	2	0	0	-	5	-	-	-	-	2.8 TB	9.7 TB	
<b>TOTAL</b>		<b>#14</b>	<b>13</b>	<b>40</b>	<b>3</b>	<b>5</b>	<b>72</b>	<b>385K</b>	<b>39</b>	<b>86</b>	<b>7%</b>	<b>8.4 TB</b>	<b>16.2 TB</b>	<b>15%</b>

Figure 8.1-b: A description of the GILDA Testbed (source <https://gilda.ct.infn.it/>).

We have used the GILDA test bed, a test environment for HEP Grid applications, to validate the results taken by the deployment of the DIANA implementation (as described in earlier chapters). GILDA facilitates testing the capabilities of Grid applications and it consists of several academic and “commercial” sites. The testbed has a series of sites and services

(including the execution environment for physics analysis) and is located in several sites in Europe and South America. Figure 8.1-b describes the testbed, its constituent sites, their load and their waiting and running jobs and storage resources on the testbed. The resources are geographically distributed and connected through a high speed WAN. All of the machines run the CERN Scientific Linux operating system. The GILDA Testbed has some of the emerging Grid-standards based EGEE applications already installed, and we made use of those components and applications to test our scheduling approaches. We used the GILDA testbed to run physics analysis applications as a proof-of-concept demonstrator.

We first submitted jobs on the GILDA Testbed without the DIANA Scheduler and measured the queue times, execution times and data transfer times. After this, jobs are re-submitted following the algorithm employed in the DIANA Scheduler which includes the measurement of queue times, execution times and data transfer times, as described in chapter 4. We took a particular computationally intensive job from the high energy physics (HEP) group CMS which produced a very large amount of data. We selected this CMS job because its execution time is of the order of minutes, in order to minimise the effect of varying network characteristics. The total time for job execution is discussed below. Tests are performed by submitting jobs through GILDA's user interface. The client machine was a Pentium based machine with a 2.4 GHz processor and 1 GB RAM. The network card was of 100 Mbps capacity.

Due to varying load conditions, it can be difficult to estimate the true effect of the DIANA scheduling approach if jobs are run at different times and/or results of various approaches are taken at different times. In order to compare the two approaches, we executed short duration jobs in as near-identical environments as was practically possible. DIANA Scheduling is equally applicable to short and long duration jobs. For the longer jobs it is the execution time which will vary and accordingly queue times will also increase. The execution cost will remain the same with time since once a job is submitted, whether it is a long or a short job, it will not be pre-empted until it completes its execution (and therefore it is not time dependant). The same is the case for the data transfer cost which should remain the same whether a longer job is being executed or a shorter job is being scheduled. The only variable which can change with time is the network cost. Although the network cost can influence the data transfer cost it does not affect the execution time since jobs do not communicate with

each other during execution. The data transfer cost is the replication cost and is equally important for the longer and shorter jobs as far as their execution times are concerned. From this we can conclude that although these results are being presented for short duration jobs they are equally applicable to long duration jobs and therefore the results can be generalized. We submitted differing numbers of jobs. Firstly we submitted 25 jobs on the GILDA Testbed and observed their queue time and execution times. The GILDA Testbed employed the gLite workload management system as a meta-scheduler and therefore the submitted jobs followed either eager or lazy scheduling with resources being allocated on a First Come First Served (FCFS) basis. Figure 8.2 and 8.3 show the queue and execution times of gLite workload management against which we are comparing the DIANA meta-scheduler. Secondly, we submitted the same number of jobs three times and re-measured the queue and execution times. Then we increased the number of jobs to 500 and then gradually to 1000, so that we could check the capability of the existing matchmaking and scheduling system. We increased the number of jobs for two main reasons: to monitor how the queue size increases over time and in which proportion the meta-scheduler submits the jobs (that is whether the jobs are submitted to some specific site or to a number of CPUs at different locations depending on the queue size and the computing capability). Then we calculated and plotted the queue times and investigated how it increased and decreased with the number of jobs.

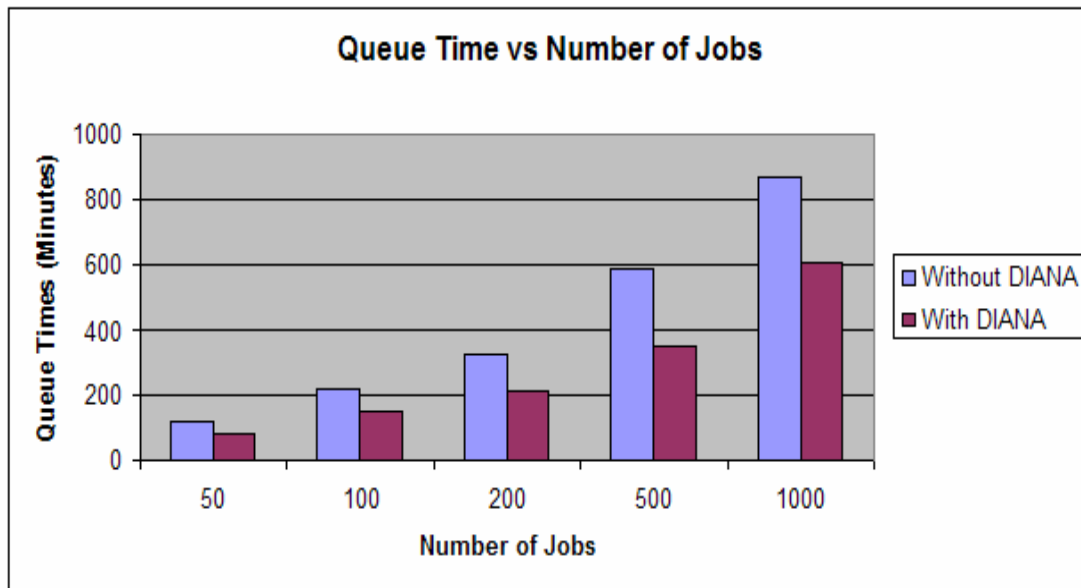


Figure 8.2: Queue time versus number of jobs.

We observe from the results presented in Figures 8.2 and 8.3 that both queue and execution times follow very similar trends. This is primarily due to the fact that DIANA preferentially selected those sites for job execution which could quickly execute jobs (i.e. those that have short local queues with low latency). The trends in figures 8.2 and 8.3 show that queue time is almost proportional to execution time because if the job is running and taking more time on the processor, the waiting time of the new job will also increase correspondingly since it will spend more time in the queue. Although the execution time does not include queue times, a higher number of jobs running at a site can influence the queue time. Furthermore, increasing the number of jobs in the queue can influence the overall job completion times (i.e. the scheduling time, queuing time and execution time) of the new jobs since they will be competing for the resources to get an execution slot, especially if the jobs are composed of sub-jobs. Large jobs are divided into small sub-jobs after a job partitioning process, and most of the time work on the same set of the data. They have similar characteristics and are treated as independent jobs during scheduling, queuing and execution stages. However, their output is returned to the user as a single aggregated unit. These subjobs are always scheduled on a single site, and the overall time of the job depends on the execution of these subjobs. Some of these jobs will be in the queue and others will be running but the overall time of execution will be the aggregate time when all these subjobs complete their execution. There is no empirical or quantitative proof of this assertion but trends in figures 8.2 and 8.3 lead us to this conclusion and this could be one possible future direction of work.

The queue time of the meta-schedulers and the local resource management systems is very significant in the Grid environment and it takes a large proportion of the job's overall time (see Figure 8.2). Sometimes this is greater than the execution time if the resources are scarce compared to the job frequency. In our experimental setup, we took only a single job queue in the meta-scheduler and we assumed that all jobs have the same priority. Multi-queue and multi-priority job scenarios will be discussed later in this chapter. In fact, the job allocation algorithm being employed by DIANA in this case is also based on the FCFS principle. The FCFS queue is the simplest and incurs almost no system overhead. The queue time here is the sum of the time in the meta-scheduler queue and the time spent in the queue of the local resource manager. The graph of the queue times when the number of the jobs changes is shown in Figure 8.2. It shows that the queue grows with an increasing number of jobs and that the number of jobs waiting for the allocation of the processors for execution also

increases. The graph shown in Figure 8.2 is based on the average values of times for varying number of jobs as mentioned previously. Improvements in the queue times of the jobs due to DIANA scheduling are also depicted in the same figure. From the figure it is clear that the DIANA scheduling technique significantly decreases the queue time of the jobs. This is because only those sites were selected for job placement which had fewest jobs in the queue and which were likely to quickly execute the jobs once scheduled on that site, were selected for job placement.

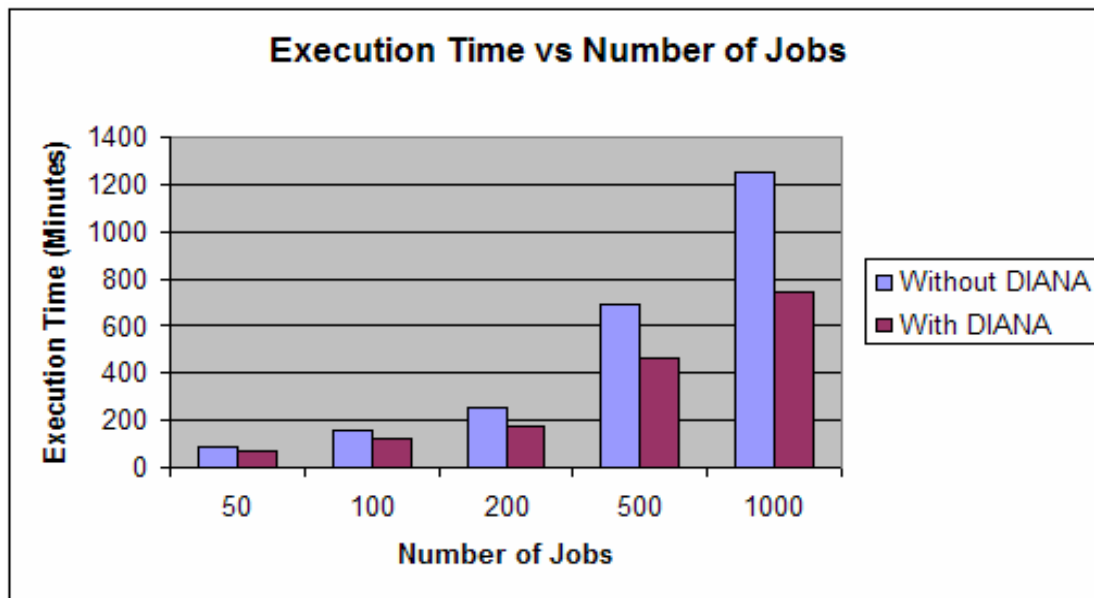


Figure 8.3: Execution time versus number of jobs.

The execution time is normally longer than the processor time consumed by the process because the CPU is doing other things besides just running the process, including running other user jobs and operating system processes or waiting for disk or network I/O. The execution time does not, however, include queue time or waiting time. By increasing the number of the jobs, it is evident from Figure 8.3 that the average time to execute a job is increased. More competing jobs clearly require more time for a specific job to complete. From figures 8.3 it is clear that the DIANA scheduling approach has improved the execution times of the jobs. This time is calculated by dividing the available computing power by the number of jobs and is indicative of the aggregated execution times. Only one job is executed on a CPU at one time, and jobs cannot run in parallel on that CPU since we are following a non pre-emptive scheduling model. More CPUs on a site can execute a higher number of

(sub-) jobs and more competing jobs clearly mean more time for a specific job to complete. DIANA has improved the execution times of the jobs since it selected only those sites for the job execution which had the required data, had less loads, had fewer jobs in the queue and all this contributed to the execution optimization. Otherwise the sites having a higher number of jobs already running or heavily loaded sites can make the execution times worse.

The graphs in Figure 8.3 show the execution optimization achieved by employing the DIANA algorithm. We can see that with an increasing number of jobs the execution performance increases which indicates the effect of the DIANA scheduling approach for the job scheduling. Here we note that the effect of DIANA became more significant as the number of jobs increased since DIANA identifies only those sites for job executions which are least loaded and which preferably have the required data, since this will reduce any transfer times. The site should also have a better network capacity to transfer the job output data back to the client side. DIANA is equally applicable to compute intensive jobs (as well as data intensive jobs) since it will find a site where there is least queue, and jobs when placed will get higher execution priorities, as explained in earlier chapters. Moreover the output of the compute operation will be quickly transferred to the submission site as a result of DIANA's optimal selection of the link between the submission and execution nodes.

Figure 8.4 shows the replica selection by the Data Location Service (DLS), which is based on the data transfer cost since DLS will select only that replica of a dataset which has the least data transfer cost. We measured the parameters required for calculating the data transfer and network costs. Packet loss and jitter of the sites was almost insignificant since the network links between the GILDA Testbed sites were rather stable and these values hardly make any difference on the data transfer and network costs. Since most of the Testbed sites are in Europe, the RTT remained almost the same other than for those sites in Brazil and China. Bandwidth was the only parameter which varied across sites and could obviously influence the data transfer and network costs and could dictate how the scheduler selected a dataset replica for the job. We took three sets of files of varying size (10, 50 and 100 GB) to demonstrate that against each required dataset, the scheduler could select a site having the shortest transfer time. From Figure 8.4 it is clear that for all three cases the shortest transfer time was for the site which had the highest bandwidth i.e. 1000 Mbps and most of the jobs used this dataset since it reduced the overall execution time of the jobs. With an increase in

the available bandwidth, the replica transfer time decreases and this remains valid for the datasets of all sizes. These measurements do not show an exact linear relationship due to delays and saturation in the network. Since we are using a public network, there can be traffic from other applications and users on the network although we tried to minimize this effect by taking the readings at times when there is very minimal activity from other users.

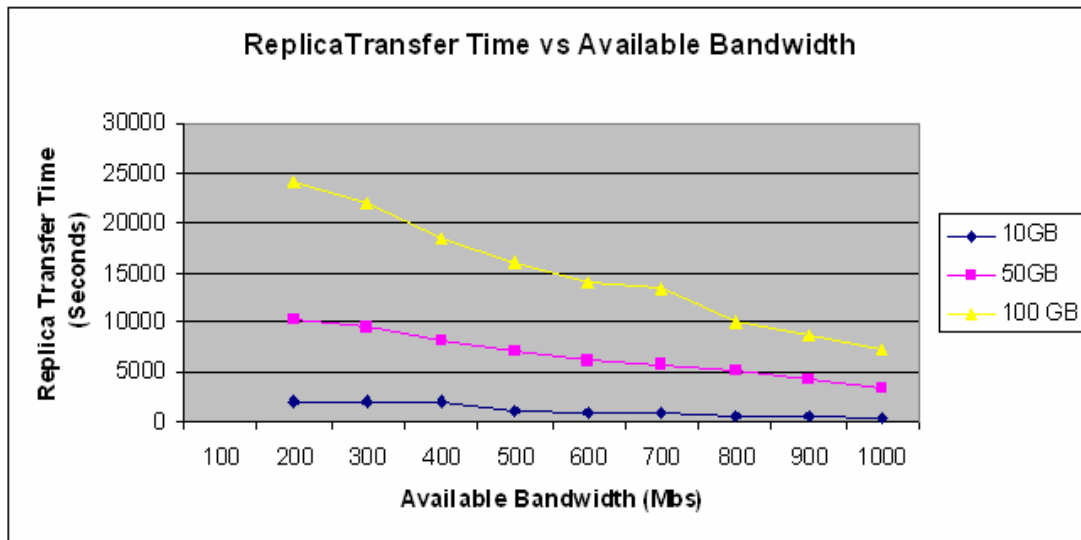


Figure 8.4: Replica transfer vs. network cost.

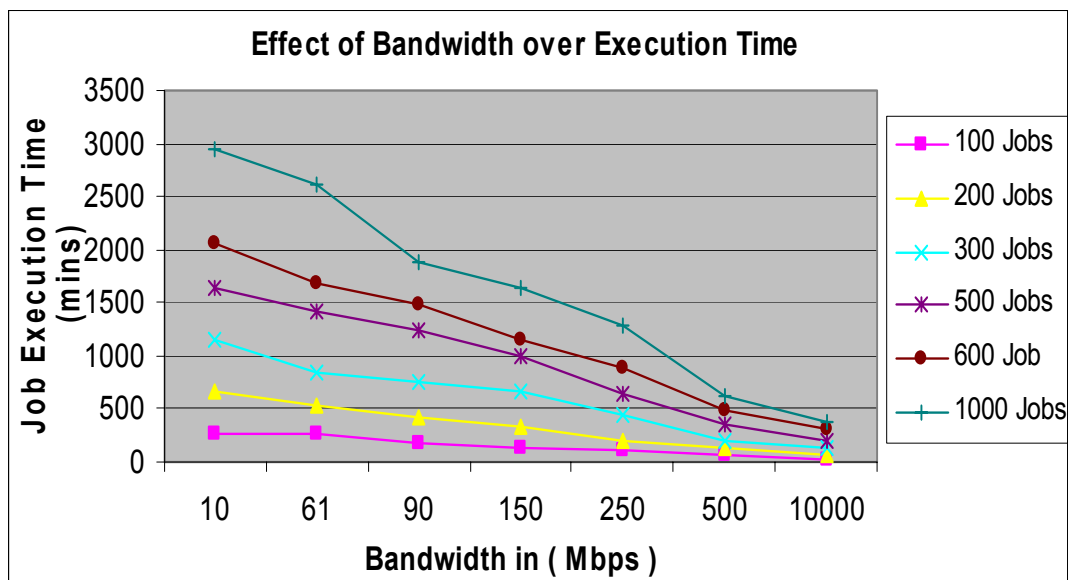


Fig. 8.5: Execution times vs. bandwidth.

Figure 8.5 demonstrates the results related to network issues which have a high impact on the execution of the data intensive jobs. In this experiment, we submitted the same number of jobs to different sites with different network conditions. The bandwidth varied from 10MB/s to 1000MB/s to enable us to gauge its effect on the job execution time. We used Iperf [170] to generate the extra network traffic and to saturate the network so that available bandwidth could vary from 10 to 1000 Mbps. In these tests we showed the effect of bandwidth on the execution time of the jobs. The data size was the same for all the jobs. Here the execution time included the time required to schedule and execute the job to one of the ‘best sites’ plus the time required in sending the data and job to that remote site and the time elapsed in queuing on that remote site. We used different networks to check the influence of the network parameters on the data transfer cost. From the comparison graph in Figure 8.5, we note that the network plays a vital role in scheduling decisions. The round-robin scheduler will schedule the job to one of the sites without consulting the network conditions of that site. This approach will cause the user additional wait time since more time is consumed in transferring the executable and the data. In our proposed approach the network and bandwidth parameters are considered to select the best sites before making any scheduling decisions for data intensive jobs and we can see the impact of this approach in Figure 8.5.

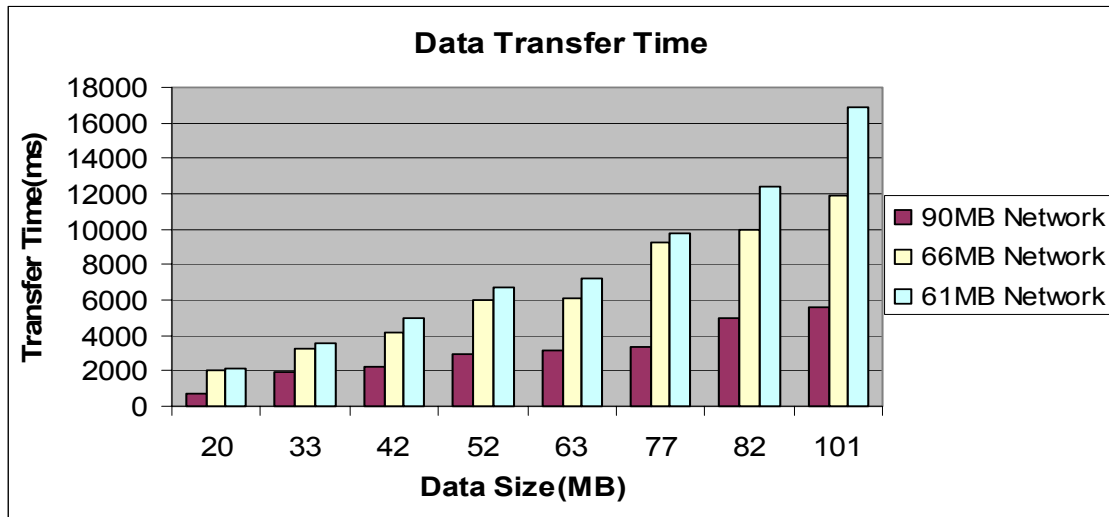


Figure 8.6: Network cost influence on the data transfer cost.

It is an observed fact that communicating over lower bandwidths often results in high network costs due to more packet losses and a longer RTT (round trip time). The increase in

network cost can also affect the overall performance of the distributed system especially in terms of transfer and communication time and therefore it is an important consideration for any scheduling decisions. A lower bandwidth results in high network costs and the increase in network cost also affects the overall performance of the distributed system. Figure 8.6 demonstrates that with increasing network capacity, the RTT and network losses decrease, which leads to reduced transfer times and hence better execution times.

In actual operating conditions for the LHC experiments (i.e. in the full production data analysis at CERN), there will be hundreds of jobs in the queue as well as in execution mode and this should allow optimization of the scheduling process with a greater factor than that shown in the graphs, since results show an upward trend with an increasing number of jobs. We can ascertain from the graphs that as the number of jobs increases, DIANA has a more profound impact on the scheduling optimization and execution of the jobs. Since most of these jobs take the data from the few selected locations where the replica of that dataset exists, it is assumed that execution and transfer times will decrease further when thousands of jobs take the actual input data from optimal locations as demonstrated in the Figure 8.5 above. In this case, the efficiency of the DIANA scheduling approach will increase further since it is better suited for environments where large numbers of jobs are involved and a great amount of data is handled. Consequently, DIANA scheduling will help to decrease the overall execution times of LHC jobs and will therefore provide an efficient way to optimize data intensive Grid jobs, although this of course needs to be verified in practice during LHC operation from late 2007 onwards.

### **8.3 DIANA P2P Scheduling Results**

We present here a performance comparison conducted using the DIANA P2P meta-scheduler which is a Web Service that uses a Grid services framework called JClarens to deploy this service. We implemented a classical scheduling algorithm which works in a round-robin manner to compare it with the DIANA P2P meta-scheduler for job scheduling on various sites. Henceforth, we will refer to it as a ‘Round Robin Scheduler’ or ‘Simple Scheduler’. For simplicity we have used our own test Grid (rather than a production environment) to obtain results since a production environment requires the installation of many other Grid components that are not required for our experiments. We used five sites located in Pakistan (NUST), Switzerland (CERN), USA (Caltech) and the UK (UWE) for the purposes of our

tests. Site 1 has four nodes, and the remaining four sites have five nodes each. Two types of jobs are used in these experiments. One type of job is compute intensive which is a simple prime number calculator (between a specified range) and the other is a data intensive physics analysis job which requires large amounts of data as input but also performs computation over this data. This section also demonstrates the capability of the DS for discovering and propagating information related to resources and services. Moreover, the impact of job steering and job monitoring on the scheduling decisions is also shown in this section.

### **8.3.1 Execution and Queue times with DIANA on the Custom Testbed**

Using the custom Grid testbed discussed above, in our first experiment, we submitted 1000 compute intensive jobs and calculated their execution times. Condor is used as a local scheduler for all of our tests. The execution times included the time required to schedule and execute the job to one of the ‘best sites’ plus the time required in sending the data and job to that remote site. The scheduling decision made by DIANA in this experiment is independent of the queue mechanism (Shortest Job First (SJF) or priority based queue) and therefore the first experiment used a single queue. As shown in Figure 8.7, it is evident that the DIANA P2PScheduler performed better than the Round Robin Scheduler. In this research, the main intention is to increase performance by utilizing Grid resources more effectively. In our scenario, DIANA has been able to select and employ those resources which are least loaded, have smaller queues and which can run the jobs more efficiently than the Round Robin Scheduler.

We then introduced the multi-queue mechanism and measured the associated queue times of the jobs (see Figure 8.8) to compare how effectively DIANA can reduce wait times. The queue time here is the sum of the time in the meta-scheduler queue and the time spent in the queue of the local resource manager. Sometimes the queue time is even greater than the execution time if the resources are scarce compared to the job frequency. Firstly we employed the Shortest Job First (SJF) queue mechanism and measured the overall execution time of the job to check the impact of the approach. In this case compute intensive bulk jobs were placed in the queue before the DIANA scheduler allocated the short jobs first to the appropriate sites. The queue was maintained on a FCFS basis. These jobs were similar with respect to their requirements (prime numbers calculation) but they were different with respect

to their inputs as each job had different input ranges. These jobs were of varying processor requirements such as 8, 17, 26, and 35 processors. The job demanding 8 processors has an input range 1-19999, 17 processors job has an input range 1-99999, 26 processors job has an input range 1-444444, and 35 processors job has input range 1-555555. All jobs were submitted to the scheduler, which arranged them in its queue in a SJF (Shortest Job First) manner on the basis of the job's processors requirement.

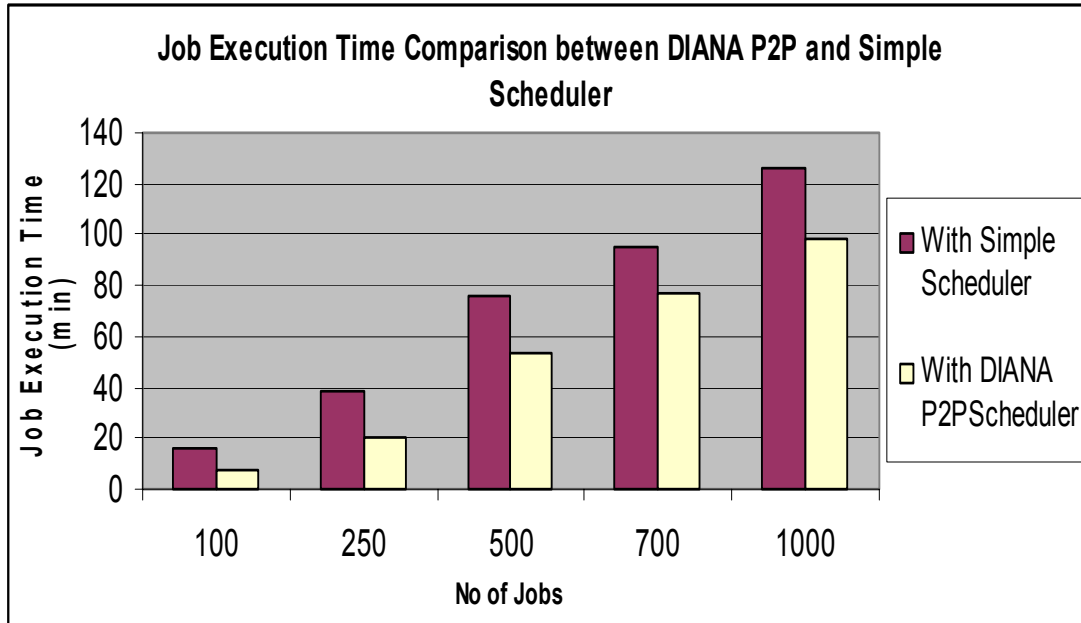


Figure 8.7: Execution time with and without the DIANA P2PScheduler.

In the comparison graph of Figure 8.8, it is clear that the performance (or execution) time of jobs obtained using DIANA was better than that of the Round Robin Scheduler. Without pre-emption, the jobs ran to completion before a new job was selected. The reason behind this was that DIANA worked on a SJF basis which reduced the execution time as short jobs did not have to wait for long jobs. As stated earlier SJF minimizes the average wait time because it services small processes before it services large ones. While it minimizes average wait time, it may penalize the jobs with large service time requests. The jobs with large service times tended to be left in the queue waiting for their turn while the small jobs received service. If the site had little additional time after servicing the short jobs, jobs with large service times would never be served. This total 'starvation' of large jobs may be a serious limitation of this SJF algorithm. Therefore this algorithm is ideal for bulk jobs which are of

short duration but might not be suitable for those jobs which are supposed to run for days or months.

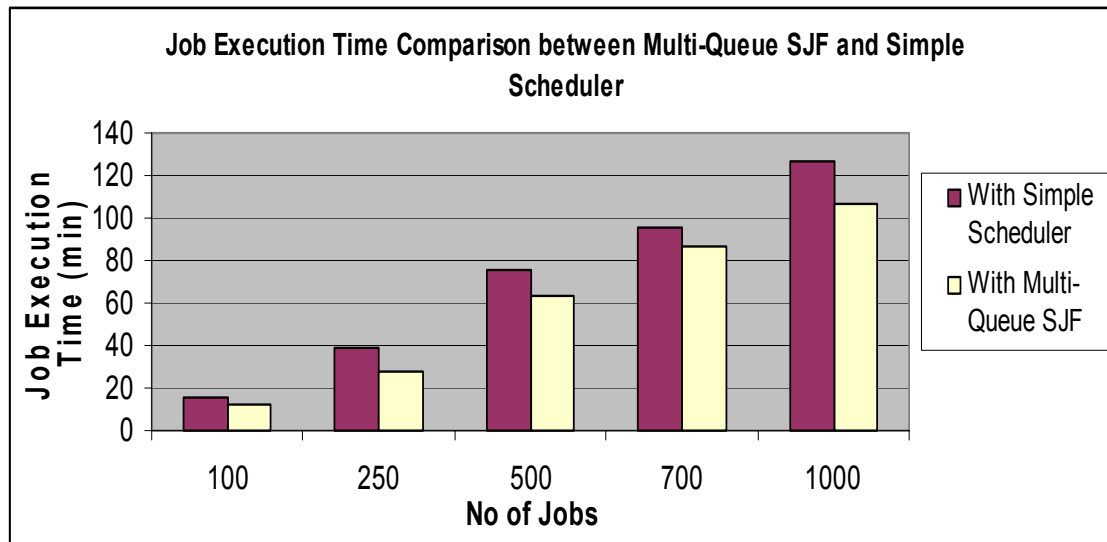
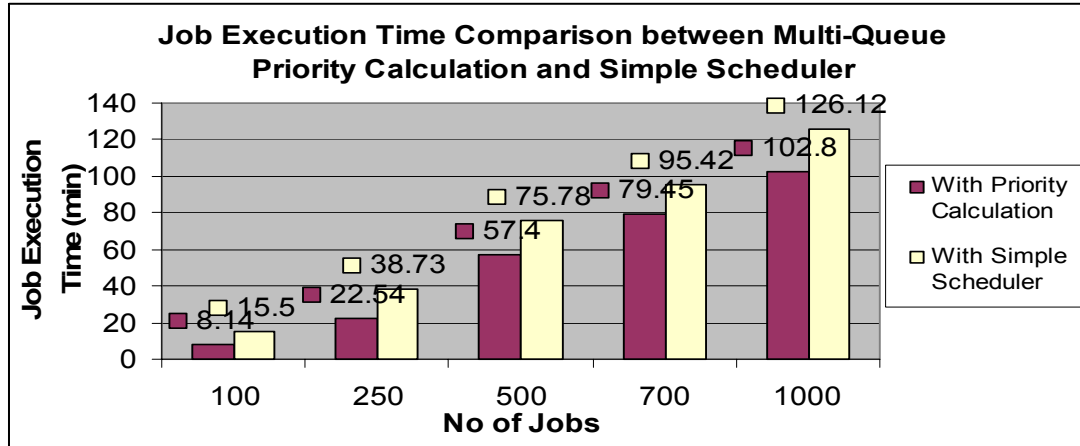


Figure 8.8: The effect of Shortest Job First on execution time.

Next we submitted a set of jobs to calculate the job execution time with a multi-queue priority mechanism. These compute intensive jobs were similar with respect to their prime number requirements but they were different with respect to their inputs since each job then had a different priority. In priority scheduling, processes were allocated to the CPU on the basis of an externally assigned priority. The Scheduler ran the highest-priority processes first and allowed CPU's to be allocated preferentially for important jobs. The key to the performance of priority scheduling is in choosing priorities for the processes. However, priority scheduling may cause low-priority processes to starve. This starvation can be compensated for if the priorities are internally computed.

Suppose one parameter in the priority assignment function is the amount of time the process has been waiting. The longer a process waits, the higher its priority becomes. This strategy tends to eliminate the starvation problem. Moreover, the SJF algorithm maintains the high priority queue in order of increasing job lengths. When a job comes in, it is inserted in the highest priority queue based on its processor requirements. When current processing has been completed, the scheduler picks the one at the head of the queue and executes it. As shown in Figures 8.9 and 8.10, the DIANA P2PScheduler with its multi-queue priority mechanism had

an improved execution compared to the Simple Scheduler. Multi-queuing not only enabled the short job first execution but also managed the queues on a priority basis and this mechanism significantly reduced the total execution times.



Fig

ure 8.9: Using multi-queue and priority calculation.

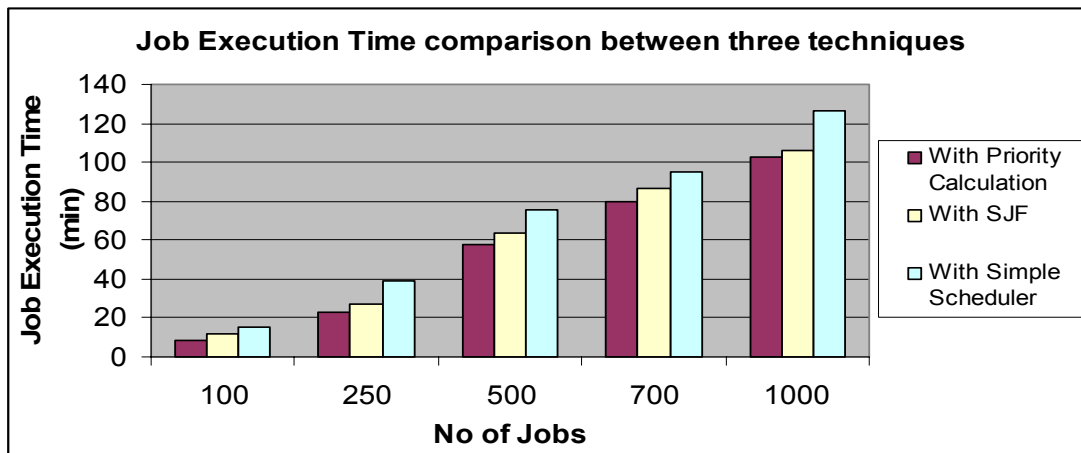


Figure 8.10: Execution time comparison.

### 8.3.2 Discovery and propagation performance in a Peer to Peer Network (P2P)

We calculated the time it takes for retrieving the data related to scheduling Peers from the Discovery Service (DS), using a java based XML-RPC client. All of these tests were conducted on an Intel P4 machine with a 2.8 GHz processor with 256 MB memory. The graphs in Figures 8.11, 8.12 and 8.13 show the results of data retrieval with varying numbers

of scheduling instances. These results were calculated both from a memory-based and two different database-based storage systems (MYSQL and HSQLDB).

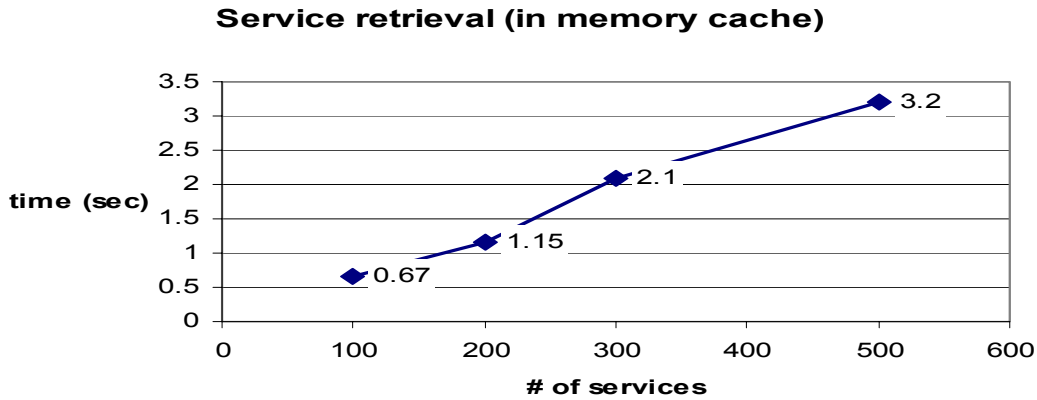


Figure 8.11: Service retrieval (in memory cache).

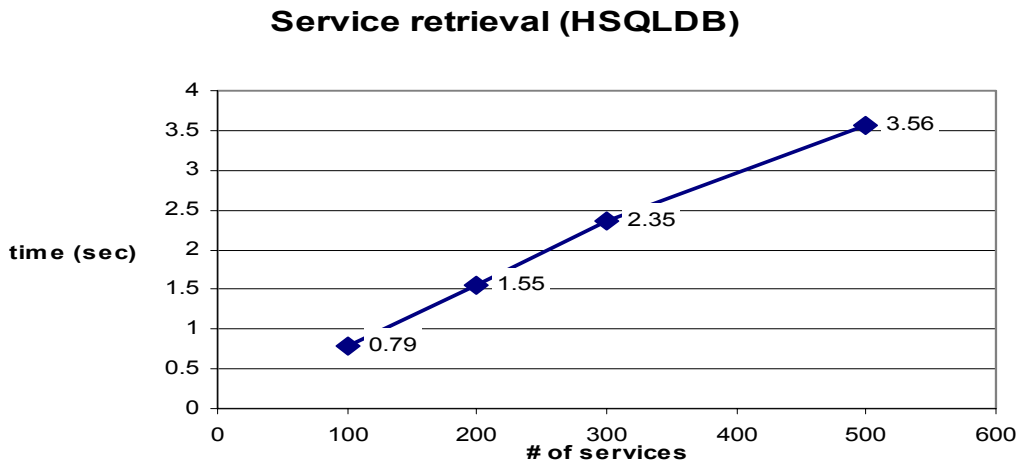


Figure 8.12: Service retrieval (HSQLDB).

The Service Descriptor contains the Endpoint list {uri, encoding (soap, xmlrpc)}, Name, Admin email, VO, uri\_suffix, Site Description, WSDL url provider\_dn, item {key, value} and the following methods (see section 6.6 for details) were used to register and access the services. These include find\_server (encoding, uri, provider\_dn, vo), find (encoding, uri, provider\_dn, vo, name) register (ServiceDescriptor[]) and deregister (ServiceDescriptor[]). The service retrieval test did not include the time taken for making the connection or for authenticating the user with the DIANA Scheduler. The somewhat larger increase in service

retrieval time as the number of instances increased is due to the overhead involved in parsing the XML-RPC response from the Clarens server. We note that the in-memory cache was suitable for fast retrieval of service data. The only problem was with the memory overflow when the scheduler registered a large number of services or data at a site or across sites. The access and propagation times of the scheduler peers and services were the same since in our implementations we used both of these metrics interchangeably. Consequently these graphs demonstrate how quickly scheduler instances discovered, communicated and propagated the scheduling related information to other peers in the Grid network. The reason we stored the peer information in the database was to ensure reliability and persistence and this came at the cost of the performance as demonstrated by Figures 8.11, 8.12 and 8.13.

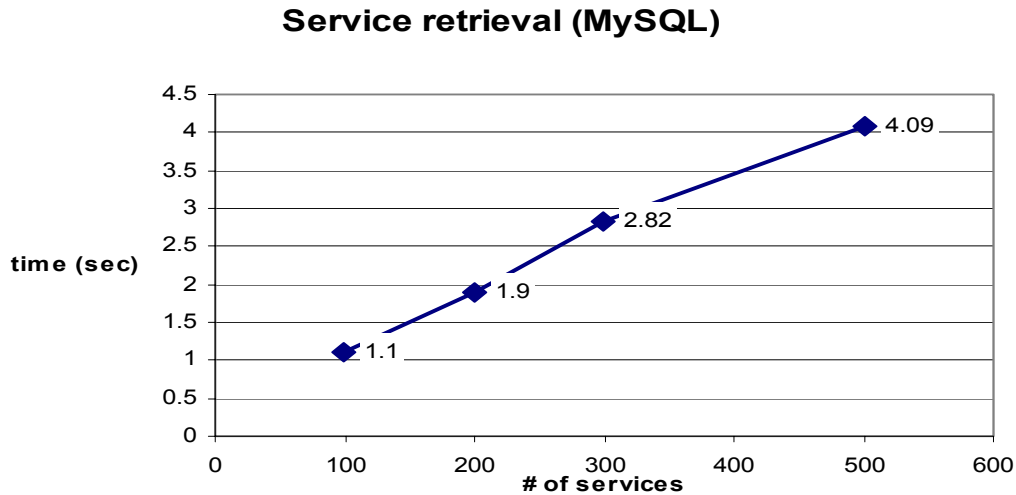


Figure 8.13: Service retrieval (MySQL).

Another test was to measure the time the DS takes to replicate node and service information to other instances of the scheduler and discovery services over the WAN. The service retrieval time was calculated based on the difference in time between services being registered at one node of JClarens (running at NUST, Pakistan, UWE Bristol and CERN, Switzerland) and becoming available at another node (running at Caltech, USA). The response we got varied from 3 seconds to 10 seconds, with it rarely going above 22 seconds. The mean of different observations was  $10.7 \pm 8.3$  seconds. The variance in the upper values is attributed to the network latency in our network. Therefore on average it took about 10.7 seconds for the transfer of one service or scheduler instance across the Grid network through

the MonALISA replication mechanism. Figure 8.14 presents the values obtained for service retrieval for a different number of attempts. These values represent the average time for a different number of attempts. The x-axis represents the number of attempts and the y-axis represents the average time taken to retrieve the service for these numbers of attempts. This is the average time rather than the time for a single attempt. We did not try to determine the actual values of these latencies and their effect on the service retrieval data this being one direction for future work. Moreover, due to the factors involved in network monitoring and performance evaluation, this becomes more a network performance measurement problem than a scheduling issue and is not the core subject of interest in this thesis.

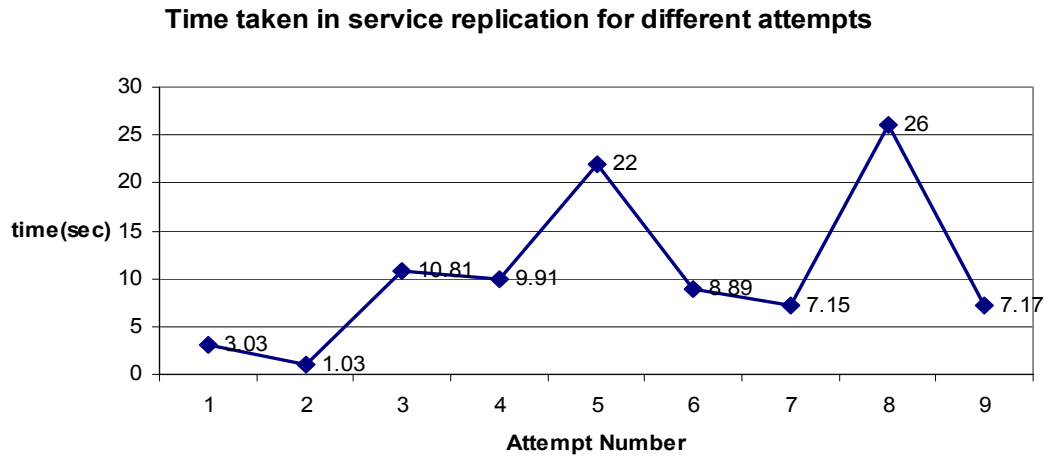


Figure 8.14: Service replication.

### 8.3.3 Job Migration and Monitoring

The DIANA Scheduler can steer jobs to sites which can ensure their optimal execution; for this purpose, the scheduler uses a steering service. Note that job migration and steering will be used interchangeably in this section. Since the steering service is providing the import and export behaviour to jobs, therefore in this section we will use the word “steer” to describe the export and import functionality. This empowers the scheduler to periodically monitor the performance of the job (using the job monitoring service), to make dynamic estimates of the job completion time, and to reschedule the job when required so that the productivity and throughput of the Grid infrastructure is enhanced. This does not mean that the job in progress is checkpointed and re-started from the previous execution state on the target site. Rather, the

DIANA scheduler uses this steering service to track those jobs which are extremely slow on a site and their re-scheduling on another site could potentially improve the execution times. The previous state of the job is not saved and migrated since this can be a very expensive operation for data intensive jobs. Furthermore, the steering service is only called when a job is closer to failure and there are very few chances of its execution completion on the current site. Otherwise a non-pre-emptive mode of scheduling and execution is strictly followed in the DIANA scheduler.

Moreover, since the APIs of the steering service also enable users to obtain this information, the scheduler can also make such rescheduling decisions when the performance is deemed insufficient. It might be difficult to get near real-time performance from the system due to the fact that it takes some time to detect any slow processing rate of a job, due to the large job queues and the fact that other jobs are running. The earlier a slow site or machine is detected, the quicker it is to export its jobs to better performing sites to optimize execution. However, once this has been detected a rescheduling of the job, based on updated monitoring information, can dramatically reduce the overall execution time since the scheduler will ensure that the jobs will get higher priorities on the target site and spend less time in the queue via this job migration.

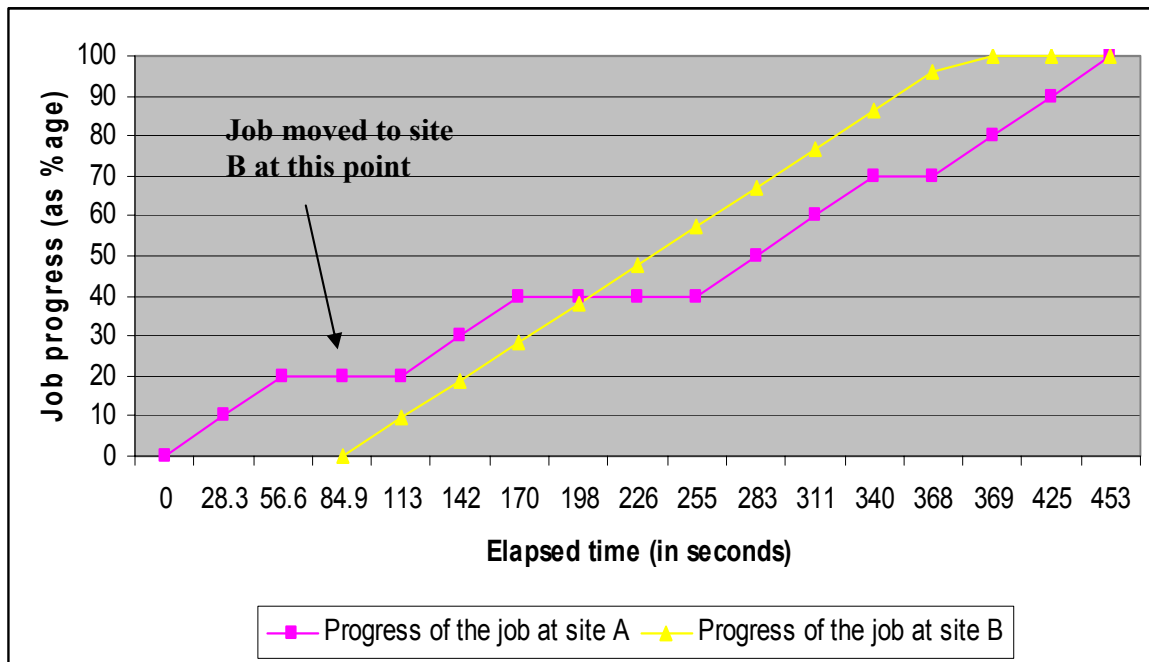


Figure 8.15: Job Completion at different sites.

Figure 8.15 shows the completion time of a job under different scenarios and demonstrates performance achievements after migrating the job to a different site. Figure 8.16 shows the estimated and actual completion time of the job. Currently this estimate is calculated by running the job many times on different machines that have negligible CPU load. This estimate comes out to be around 283 seconds and therefore we assume that if a CPU is available (and not loaded) the job will normally complete in around this time. Moreover, Condor provides us the ability to see how much "wall-clock time" the job has actually accumulated on a node or in a Condor pool. Note that this "wall-clock" time does not include the time the job consumes while it is idle and waiting for the CPU or other resources to become available. We used this feature of Condor coupled with the above-mentioned assumption as a way to measure the progress of the job when it is running on a node with significant CPU load. Thus, if the job has accumulated 141 seconds of wall-clock time (as shown by Condor) when it is running on such a node, we assume that roughly 50% of the job is complete, even though the time elapsed since the job had been scheduled on that node may be greater.

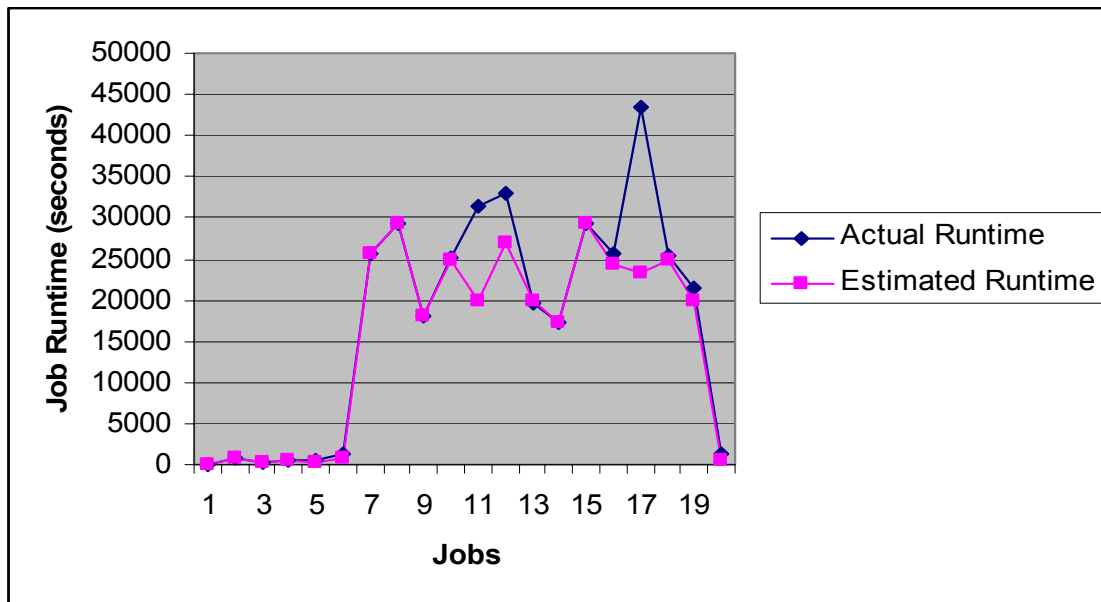


Figure 8.16: Actual & Estimated Runtimes for 20 test cases.

Based on this data, we chart the progress of a job from 0 to 100% (as shown in Figure 8.15) while it was waiting on a node A with significant CPU load. The purple line shows the job that was running on site A under significant CPU load. The scheduler monitored the progress

of this job (using the job monitoring service) and decided to move this job based on its slow execution rate (note that the user could equally have moved the job from site A to site B manually). This job was then rescheduled on some new site B, while the job was also allowed to continue running on site A for testing purposes. It is clear that after rescheduling, the job has completed much sooner than the copy of the job that was executing at site A (as indicated by the yellow line).

The job can be completed even quicker than the recorded time of 369 seconds if it is checkpoint-able and flocking (job migration) [61] is enabled between site A and Site B. However, we are using non pre-emptive scheduling due to the data intensive nature of the jobs and this binds a job to continue execution once a CPU is allocated to the job. The execution gains are achieved mostly due to the queue time improvements. A critical factor that affects the job completion time is the time at which the decision to move the job is taken. The quicker the decision is taken, the better the chance that it will complete quicker. Another important factor is the time taken to transfer the data files needed by the job. All of these factors are taken into account when deciding whether a job should be migrated or allowed to run to completion at its primary location.

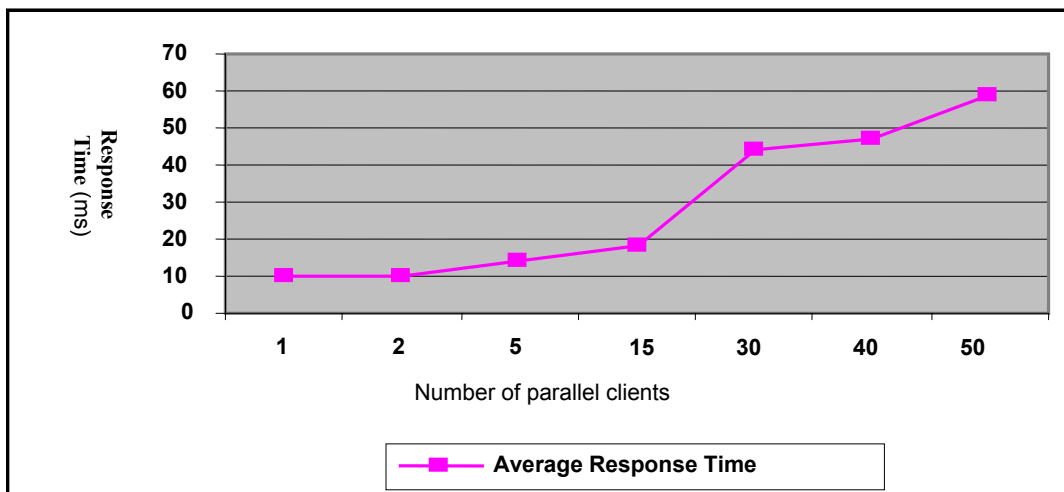


Figure 8.17: Response times for queries to Job Monitoring Service.

We also carried out a number of tests to measure the performance of the Job Monitoring Service in the scheduler. The job movements and scheduling decisions are dependent on the monitoring information and how accurately it is being measured and how quickly it is being propagated. A number of clients were initialized in parallel to call various methods of the

service to access job monitoring information. Figure 8.17 shows the results in terms of the average time taken to fulfil a request when different numbers of clients tried to access the service concurrently. The results show that the performance of the service scales well with an increasing number of clients, which means that the scheduler can monitor a large number of jobs. The client authentication process can take time therefore the client has to wait for a certain interval before getting the information.

#### **8.4 Results for Bulk Scheduling**

We present here results from a set of tests which have been conducted with the DIANA scheduler using MONARC [63] and SimGrid [45] simulations to check the algorithm behaviour for bulk scheduling. SimGrid is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. The specific goal of the SimGrid toolkit is to facilitate work in the area of distributed and parallel application scheduling on distributed computing platforms, ranging from simple network of workstations to Computational Grids. SimGrid provides several programming environments built on top of a simulation kernel. Each environment targets a specific audience and constitutes a different paradigm. Testbeds were generated from 50 nodes to 1000. Each Node had a processing power randomly generated from 10 KFLOPS to 1 GFLOPS Each Node had network connectivity from 56 kbps to 10 MB/s (maximum allowable 2 Gps).

MONARC is a simulation framework whose aim is to provide a design and optimization tool for large scale distributed computing systems. Its goals are to provide a realistic simulation of distributed computing systems, customized for specific physics data processing, and to offer a flexible and dynamic environment to evaluate the performance of a range of possible data processing architectures. The present version comes with a set of new features, such as the possibility to simulate data replication and distributed scheduling, and with more ways to represent the simulation results, in order to adapt better to the users' requirements.

In the bulk job scheduling, hundreds of thousands of jobs are submitted and once scheduled on a site, they can take days and even months to complete execution. Since we have to conduct a number of experiments with differing numbers of jobs, huge data transfers for input and output results are involved, scheduling and execution operations take a much longer time and such tests also practically consume nearly all the available resources leaving

little space for other jobs or operations. Therefore we have employed the simulation results to investigate and support the conclusions. Further, due to policy and quota enforcements on a number of our sites, jobs might not be able to be executed on the best sites and this can lead to less accurate results. Therefore we need to simulate the ideal behaviour of the scheduler to overcome any experimental issues over the testbed. Note that we conducted these tests in almost ideal circumstances with little bias from the environment and this can be quite different to the actual Grid deployment where latencies, queuing and scheduling delays occur and job failure rate can sometimes be high. Therefore this aspect of the results should be considered when interpreting the simulation results.

#### **8.4.1 MONARC Simulations for Bulk Scheduling**

The simulation setup is created to overcome the limitations of the experimental testbed for the bulk jobs. This testbed has five sites in different regions of the world. Sites have different computing capacity, bandwidth and data availability to emulate a natural Grid testbed which has a mixture of strong and weak sites. For all sites in the MONARC based testbed, each CPU has 512 MB of memory and processing power 100 (SI95). The processing time for the jobs is normally distributed, with an average of 3 hours; each CPU can execute only one job at a time (due to memory limitations). The number of processors in each site varies from 10 to 100, which means more than enough CPU power to process the jobs. Jobs can be submitted to any of the sites since DIANA functionality is available for each site in the testbed. Each site can import and export the jobs based on its load, computing capability, data availability and the network conditions. This simulated Grid testbed is created in such a way that these sites are inter-connected through high speed internet links so that job movement and data replication is encouraged for scheduling and execution optimization. Each site can execute jobs equal to the number of processors it has and the remainder of the jobs are placed in a queue. Therefore the job threshold on each site is different. This is the maximum execution capacity of a particular site i.e. the threshold beyond which jobs will be exported to remote sites for execution. This is quite different to the user job threshold which is a virtual organisation (VO) wide entity. The total number of all the jobs in the whole VO from a single user should not cross a particular limit this being controlled by the user threshold. Again this varies from one Grid deployment to another and the priority of the jobs is controlled by controlling this threshold as detailed in chapter 4.

The number of jobs that can run simultaneously on a site is equal to the number of available CPUs on that site and jobs are exported only in exceptional circumstances when the jobs in the queue are estimated to take greater times than that when exported to a remote site. As discussed in the bulk scheduling algorithm in chapter 4, the submitted jobs are divided into the subgroups of jobs which were generated through a splitting process. In the bulk scheduling, either subgroups are scheduled instead of individual jobs or the whole of the bulk is scheduled. The size of each subgroup varies in different scheduling scenarios. As the number of resources and sites increases, the size of the subgroups also varies, as now the sites can handle the subgroups of different size. This can lead to more flexible scheduling alternatives, eventually leading to scheduling optimization. When a site exports a job, the whole subgroup is exported in bulk scheduling since all the jobs in a subgroup have the same input and output requirements. All the jobs in a subgroup execute on the same set of the data and therefore it is natural to export them as one unit. The same is the case for the import of jobs. When some site requests a job import, in the case of the bulk jobs the whole subgroup is imported. This import and export phenomenon of the bulk jobs and their subgroups reduced the data transfer cost and selected the computationally optimal sites and therefore, significantly contributed in the scheduling optimization.

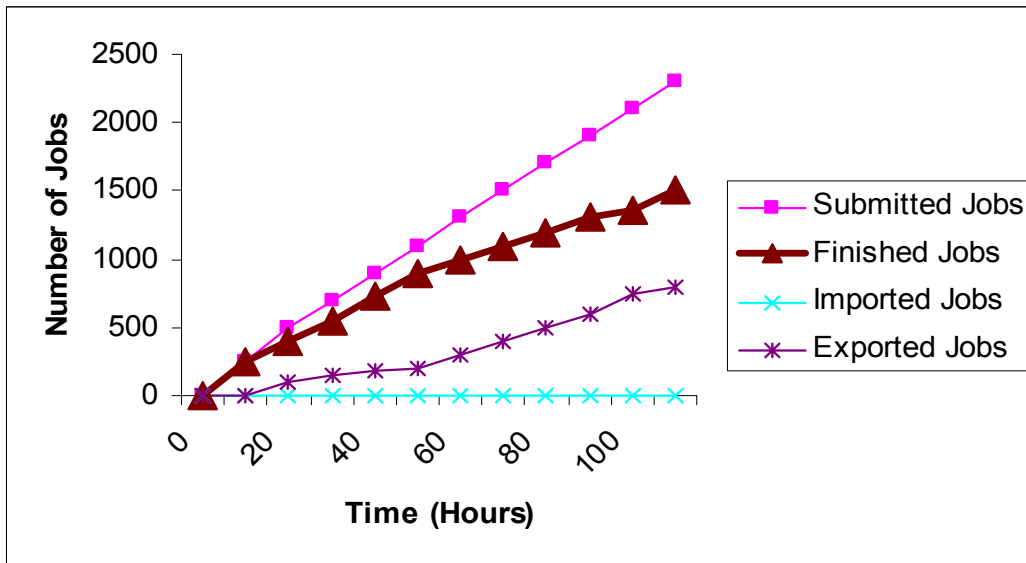


Figure 8.18: Job frequency higher than the execute capacity of the site.

First we submitted a number of jobs which exceeded the processing capacity of the candidate site and observed large queues of jobs which could be processed in an optimal manner. The bulk scheduling algorithm discussed in chapter 4 was employed to schedule the bulk jobs and the job migration algorithm was used to migrate the jobs to other sites. We discuss here the summary of the jobs which were executed locally and that of those migrated to other sites. The results suggest that as the number of jobs increased beyond the threshold limit, more and more jobs were migrated to other less loaded sites over time, since the current site selection was no longer optimal. While selecting a single site, we used DIANA so that all the network, compute and data related details discussed in chapter 4 were brought under consideration before job placement on the selected site.

Figure 8.18 shows that when the job submission frequency was much higher than the site consumption rate, the site kept on processing jobs at a constant rate, and the rest of the jobs were exported to other optimally selected sites. It is even possible for a site to export jobs which do not have the required data locally. It should be noted that a site continued processing jobs and at the same time its scheduler also migrated other jobs to more optimal sites, according to data availability and job priority. Moreover, this site simultaneously allowed the importation of jobs from other sites which required data that was available only at this site and allowed the exportation of jobs which could get better execution priority or shorter queues on remote sites. As stated earlier, we employed a non pre-emptive approach in our scheduling algorithm, and once a job started execution we could not move it. The non pre-emptive approach was followed because it avoids check-pointing and re-starting which are very expensive operations in data intensive applications. In migrating previously started jobs the whole input data plus the executable has to move with the job and this can considerably degrade the overall system performance.

Once the number of jobs at a site exceeds the threshold limit, the bulk scheduling algorithm once again uses the DIANA algorithm to select the best alternative site for execution in terms of computation power, data location, network capacity and queue length as discussed in chapter 4. As the number of jobs increases beyond the threshold, the bulk scheduling algorithm employs the policies and priorities detailed in chapter 4 to provide the desired QoS to all or to some preferred users and also restricts certain users making monopolistic decisions, thus avoiding starvation for other users. In Figure 8.18 we can see the effect of

jobs exceeding the execution capacity of a site and how jobs are exported to least loaded sites to optimize the execution process. Even the fluctuation in the submission rate is reflected by the corresponding export and execution rates. If the number of jobs being processed at a site is less than its execution capacity, then this site can import jobs from other sites in order to reduce the global execution and queue time of jobs across the whole Grid as shown in Figure 8.19.

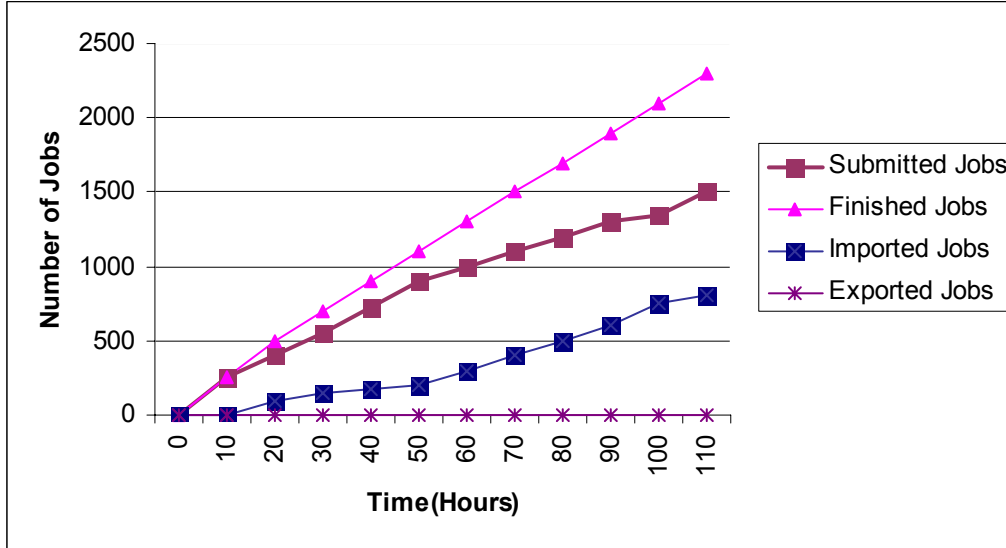


Figure 8.19: Capacity of the site greater than the submitted jobs.

In another experiment, 15 users submitted 450 jobs each. The job size was the same for all the users however, their respective quota was different. User IDs along with their respective quotas are given in figure 8.20-a.

User ID	No. of jobs	Job size (No. of processors required)	Quota
User1	450	2	5000
User2	450	2	2000
User3	450	2	1000
User4	450	2	700

User5	450	2	300
User6	450	2	1300
User7	450	2	1700
User8	450	2	900
User9	450	2	2600
User10	450	2	1800
User11	450	2	10
User12	450	2	100
User13	450	2	0
User14	450	2	10000
User15	450	2	80

Figure 8.20-a: Quota and Priority

The results suggest that the quota has a significant effect on the priority of the user and the priority of job is high when the quota value of the user is high. Figure 8.20-b shows that with increasing the value of quota, the job priority increases and vice versa.

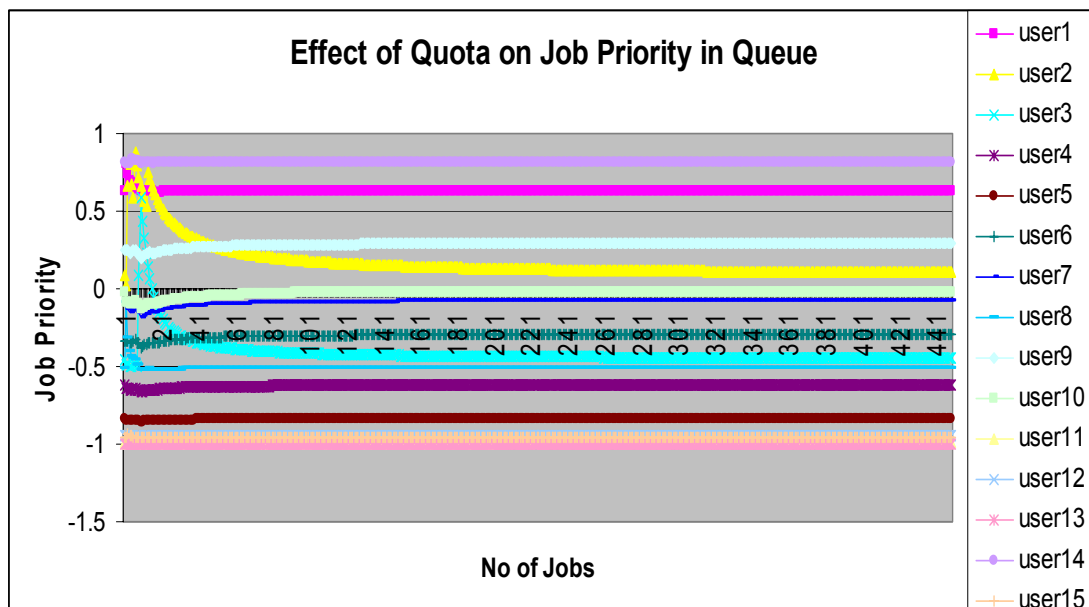


Figure 8.20-b: Quota and Priority

Figure 8.20-b is hard to understand due to the large number of users and jobs. Therefore, we reduced the results to 5 users with 50 jobs. Figure 8.20-c is shown below and it is much more visible and understandable. The quota values are already provided in the table above. In the next experiment, 10 users submitted different number of jobs in the queue. However, this time the quota and job size were the same for all users. User IDs along with the total number of jobs by each user are provided in the figure 8.21-a.

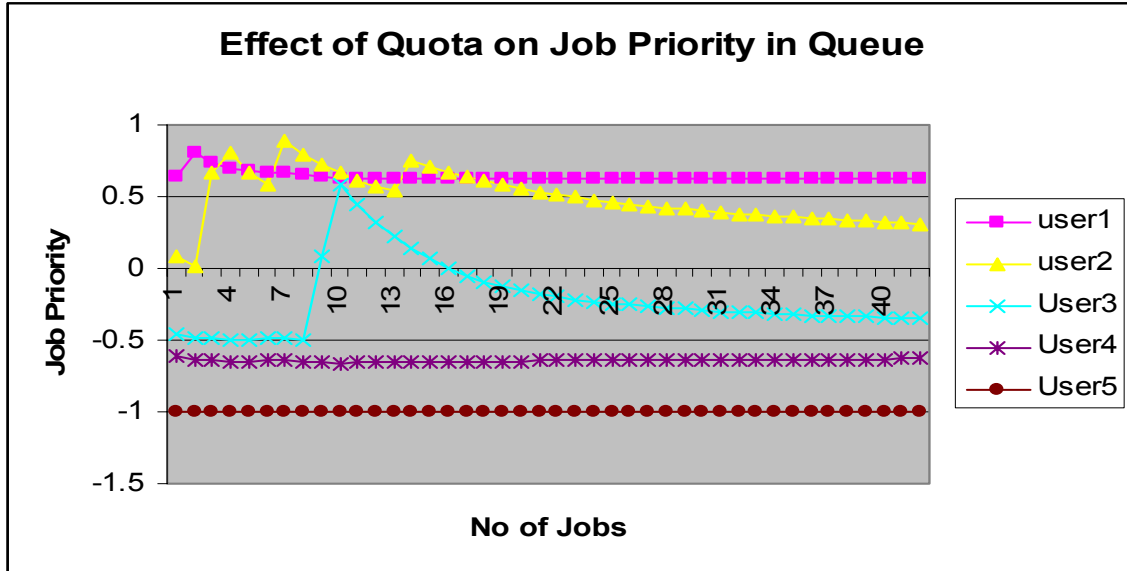


Figure 8.20-c: Quota and Priority

User ID	Quota	Job size ((No. of processors required))	Number of Jobs
User1	100	2	100
User2	100	2	50
User3	100	2	300
User4	100	2	1000
User5	100	2	500
User6	100	2	700
User7	100	2	200

User8	100	2	2
User9	100	2	1500
User10	100	2	20

Figure 8.21-a: Number of Jobs and Priority

The results suggest that job priority decreases gradually when a user keeps sending jobs in the scheduler queue. Thus, in the case when a user bombards a site with hundreds of jobs, the scheduler decreases the priority of the jobs of that particular user. Figure 8.21-b shows that at the outset, when a user starts sending his jobs in the queue its priority was high however, if he bombard the site with jobs, the priority decreased gradually with an increasing number of jobs.

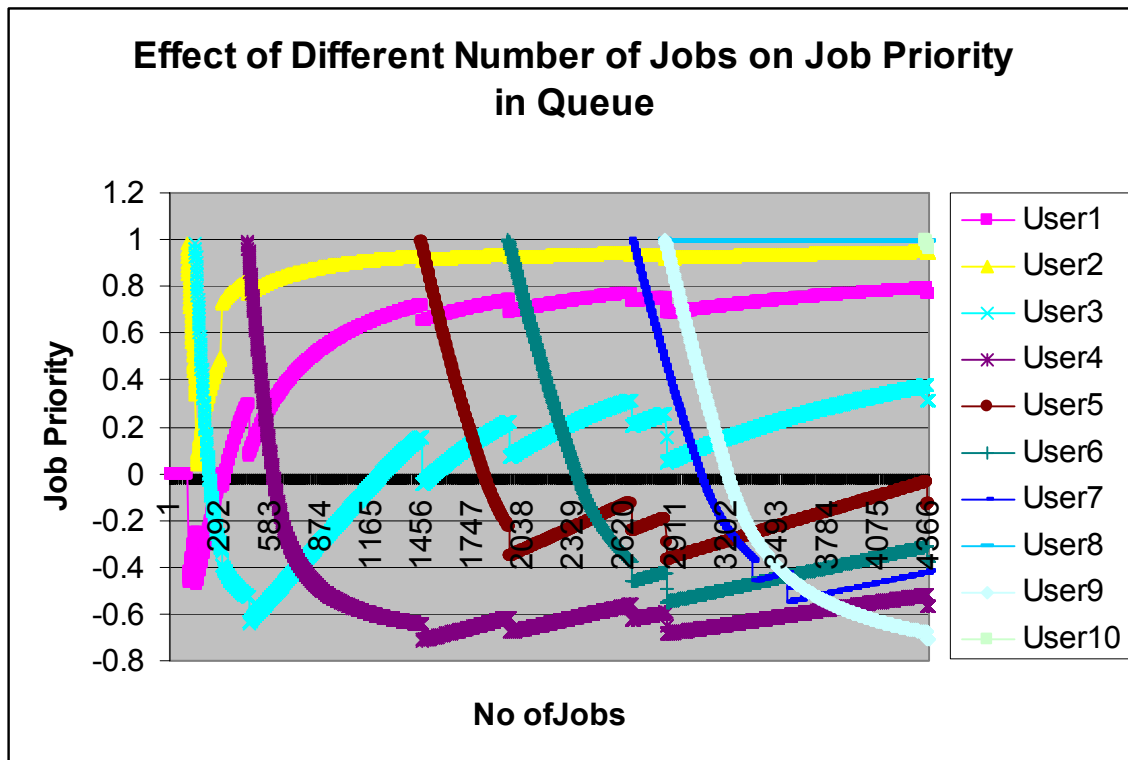


Figure 8.21-b: Number of Jobs and Priority

In the next experiment as shown in the figure 8.21-c, each of the 10 users submitted 500 jobs. All the users had a same quota value but the job size or the number of processors required for each job was different. The users along with their respective job information are detailed in the figure 8.21-c.

User ID	Quota	No. of Jobs	Job Size (No. of processors required)
User1	100	500	2
User2	100	500	5
User3	100	500	8
User4	100	500	11
User5	100	500	15
User6	100	500	20
User7	100	500	18
User8	100	500	25
User9	100	500	1
User10	100	500	30

Figure 8.21-c: Job size and Priority

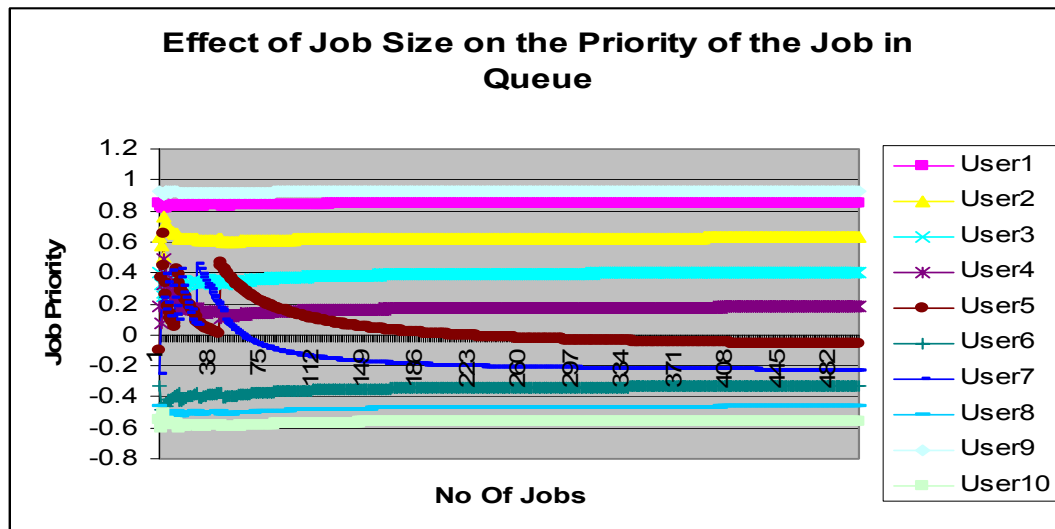


Figure 8.21-d: Job size and Priority

The results in figure 8.21-d suggest that a large number of jobs from a user lead to a lower job priority. Thus the algorithm makes best use of the available resources to schedule the short jobs before the resources are allocated to the large jobs.

#### **8.4.2 Scalability and Consistency Tests**

Next we compared the DIANA scheduling approach with two other approaches being used in common scheduling applications and the simulation results are presented here for a large number of bulk jobs and hundreds of compute and data nodes. In the following sections, first we introduce the approaches with which DIANA was being compared and other approaches which will be used during the description of the results and then comparisons are given for compute, data and network capability of all the three approaches. The results in this section are using the SimGrid environment and a summary of all these tests is given in the figure 8.22-a.

The Round Robin and the FLOP are the two other approaches against which DIANA is being compared for these bulk scheduling tests. The Round Robin approach as used in some Grid Middleware, for example Chimera VDT, takes into consideration only the list of nodes, and assigns tasks to them one by one. The Round Robin algorithm which has been implemented for these comparisons is not pre-emptive, as is the case in operating systems. With respect to the Grid Round Robin mentioned here, it is similar to a circular linked list. Each cell represents a node of the Grid and the scheduler starts from the first element in the list, schedules the incoming jobs to it, increments the list count and schedules the next job to the new node. The Round Robin algorithm is ideal for environments which have homogenous hardware resources, or clusters which have equivalent computer power. This is certainly not the case when it comes to large scale Grid systems where individual machines and sites may have many times the storage, network and computing power compared to other nodes and sites. Thus the Round Robin algorithm is expected to under perform in such environments.

The FLOP based approach used in Cluster middleware, for example Condor, considers the computation capability of the nodes. FLOP (Floating Point Operations per Second) is a common measurement for the computational capability of a computer. The FLOP based algorithm could be considered as being completely opposite to the Round Robin algorithm, because it tries to gain complete knowledge about the current state of resources, so that it can

schedule jobs to the most powerful available machine, guaranteeing the quickest possible runtime. The FLOP based algorithm could be considered wasteful, because the job which is currently being scheduled may not need the most powerful computer at all and could be processed in a machine with some moderate capabilities, with similar runtime. However FLOP based algorithms are popular amongst cluster level middleware. The central manager in such systems, on every job submission, tries to compare the capabilities of each node in the cluster, via resource descriptions called ClassAds and the node with the most capable ClassAd is selected to execute the job.

Application Type	Resource Brokering Algorithm and execution times		
	Round Robin/s	FLOP Based/s	DIANA/s
Compute Intensive (1KFLOP)	Average: 0.88 Floor: 0.49 Ceiling: 1.68	Average: 0.57 Floor: 0.27 Ceiling: 2.17	Average: 0.25 Floor: 0.21 Ceiling: 0.3
Compute Intensive (1MFLOP)	Average: 345.47 Floor: 26.19 Ceiling: 1386.29	Average: 10.45 Floor: 0.73 Ceiling: 73.52	Average: 64.1 Floor: 6.74 Ceiling: 353.2
Compute Intensive (1GFLOP)	Average: 345149.43 Floor: 25787.6 Ceiling: 1386000	Average: 59250.52 Floor: 6831.68 Ceiling: 236855	Average: 64026.55 Floor: 6643.16 Ceiling: 353107
Communication Intensive (1KB)	Average: 0.88 Floor: 0.49 Ceiling: 1.68	Average: 0.57 Floor: 0.27 Ceiling: 2.17	Average: 0.25 Floor: 0.21 Ceiling: 0.3
Communication Intensive (1 MB)	Average: 544.86 Floor: 281.65 Ceiling: 1043.15	Average: 415.83 Floor: 124.61 Ceiling: 2044.49	Average: 100.39 Floor: 99.7 Ceiling: 106.23
Communication Intensive (1 GB)	Average: 544709.35 Floor: 281539 Ceiling: 1043040	Average: 415678.4 Floor: 124433 Ceiling: 2044360	Average: 100242.02 Floor: 99505.6 Ceiling: 106129
Hybrid (1 KFLOP & 1 KB)	Average: 0.7 Floor: 0.5 Ceiling: 0.95	Average: 1.2 Floor: 0.27 Ceiling: 4.16	Average: 0.25 Floor: 0.21 Ceiling: 0.3
Hybrid (1 MFLOP & 1 MB)	Average: 546.6 Floor: 382.43 Ceiling: 755.23	Average: 1024.93 Floor: 152.01 Ceiling: 4037.47	Average: 100.91 Floor: 99.61 Ceiling: 109.58
Hybrid (1 GFLOP & 1 GB)	Average: 546475.3 Floor: 382305 Ceiling: 755036	Average: 1024.93 Floor: 152.01 Ceiling: 4037.47	Average: 100.91 Floor: 99.61 Ceiling: 109.58

Figure 8.22-a: Comparison of different algorithms with DIANA

The DIANA Algorithm considers network connectivity, data location and computational capability when scheduling. It is a variant of the simple FLOP based algorithm. FLOP based algorithms are popular in cluster level, however when it comes to the Grid, the network is hard to ignore. Clusters are usually linked via high-speed network links, often these high-speed networks are only slightly slower than the internal computer speed. Thus cluster middleware completely ignores the network, and treats each node just as an additional CPU to the host machines. This is in stark contrast to the Grid. Although some contemporary Grids are linked via high speed networks, Grid middleware often takes high speed networking for granted and does not treat it as a resource. However when it comes to a scheduling system which tries to connect the resources spread all over the world for complex compute and data operations, we cannot take the network for granted, and it has to be treated as a resource in the same manner as traditional computing resources, such as CPU and memory. Therefore to accommodate the network in resource scheduling decisions, we add one more parameter, to the normal FLOP based scheduling in these tests, the network connectivity, as measured in terms of bandwidth, between two nodes. When a job is submitted to the Grid, the resource broker requests members of the virtual organization for their current resource state, the virtual organization consisting of all the nodes which are capable of executing the application. The member nodes respond to the request by dynamically creating resource descriptions and additionally at runtime try to determine the current “real” bandwidth from the site which requested the descriptions.

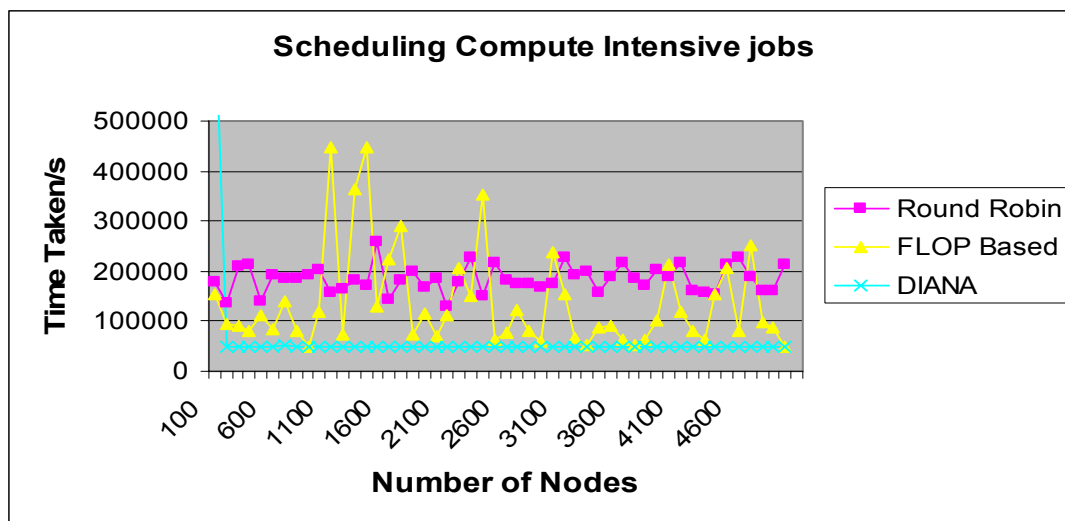


Figure 8.22-b: Node Selection in DIANA

The following two tests evaluate the efficiency of the algorithm in terms of picking suitable nodes for jobs. An algorithm not performing well in this simulation shows that it is not the best suited algorithm for data intensive scheduling and it is not able to peruse the Grid infrastructure adequately. In this test a batch of 100 compute intensive jobs was dispatched to the Grid. We took two sets of results with different processing requirements and each job had a processing requirement of 100KFLOPs and 1MFLOP respectively in each case. In this test each job's communication traffic was 10Kb. In the first test, a number of computational intensive jobs were launched to the Grid to measure which algorithm took the least time to execute the entire batch of jobs. In Figure 8.22-b, we can see that each algorithm has a lot of variation; however in all cases the DIANA algorithm shows the best performance. It is scalable for a large number of nodes and has minimum execution times when compared to other algorithms as show in figure 8.22-b. The flat line in the figure shows that DIANA is consistent and stable for a number of nodes and jobs and has the best performance when compared against other algorithms. To further diagnose the performance, a number of computationally intensive jobs were submitted to the Grid to measure which algorithm took the least time in executing the entire batch of jobs. We can see a definite pattern in figure 8.23. FLOP and DIANA behave consistently and reliably, whereas Round Robin has far too much jitter and thus is bad from a QoS perspective. DIANA performs better among all of the three and practically has no jitter. This suggests that DIANA is best suited for the scheduling of the compute intensive jobs involving some data transfer. The reason behind selecting the low communication traffic cost was to reduce the simulation overhead without affecting the authenticity and quality of the results. The reasons behind the improved performance of DIANA Scheduling include its capability to take into account the network characteristics which empowered the scheduler to select resources that could transfer the data quickly. Furthermore, DIANA Scheduler scheduled the jobs on those resources which had minimum queue and a better computing capability. These are the reasons due to which the DIANA scheduler performed well for a large number of jobs and resources and consistently outperformed the other algorithms in terms of the performance and throughput. This behaviour remains valid from a small to large number of jobs and resources, making DIANA Scheduling a suitable approach to cater the scheduling needs of diversified number of users and applications.

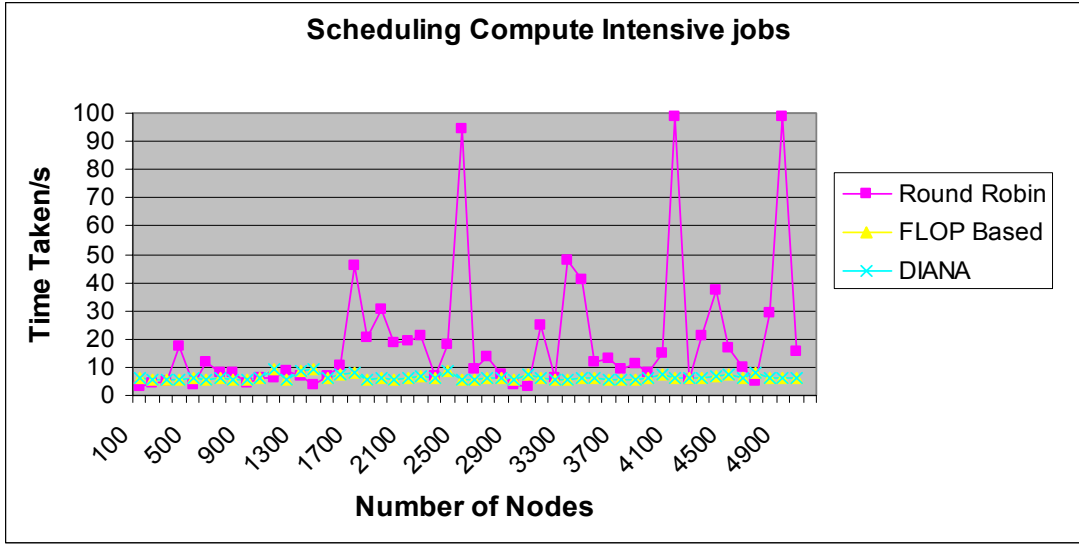


Figure 8.23: Performance comparison of the algorithms

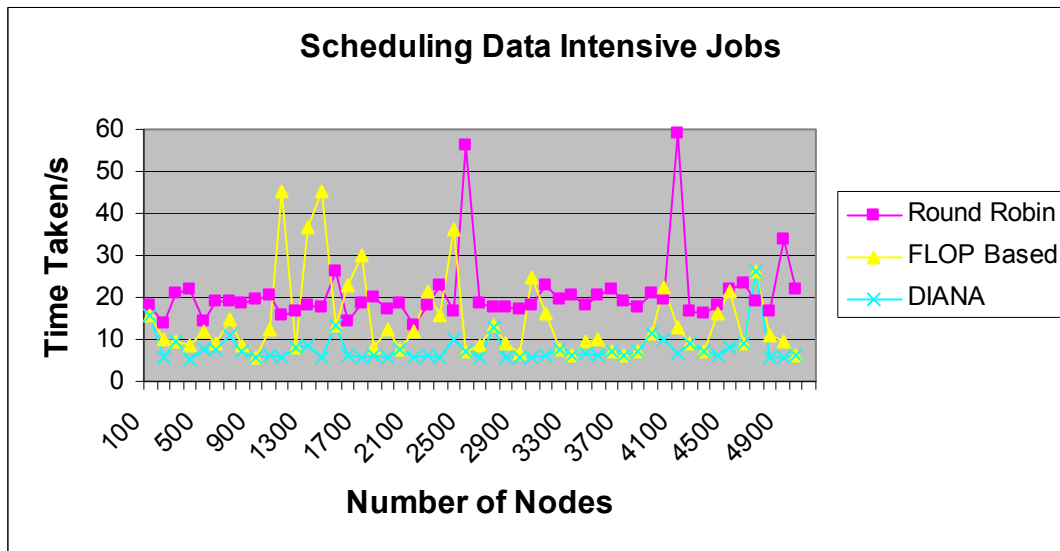


Figure 8.24: Network Efficiency of DIANA

Now we present simulation results to test the network efficiency of the DIANA algorithm. In a Grid environment nodes are connected via variable network links; an algorithm which does not use the network as a resource in the scheduling decisions will be inefficient and will waste resource as well as complete tasks in an untimely manner. The network is an especially critical resource for data intensive scheduling. As has been established in earlier chapters and through practical results: ignoring the network can lead to suboptimal scheduling decisions. In these tests 100 100KFLOP jobs were submitted with various data communication

complexities. In the first such simulation, jobs which have low communication complexity were scheduled to the Grid (100KB traffic). The response of each algorithm is shown in figure 8.24. DIANA clearly has stable and consistent performance, whereas Round Robin and FLOP show a lot of variation. This suggests that DIANA is a very suitable technique for the data intensive jobs and its consistent performance further verifies the suitability of this approach.

To evaluate further the network related aspects of DIANA and to compare it with other approaches, simulation jobs of higher computation complexity were scheduled. In this particular test jobs of 1MB were scheduled on the Grid. As shown in figure 8.25, DIANA clearly has stable and consistent performance, whereas Round Robin and FLOP show significant variation. This also suggests that as the data size increases, the optimization achieved through DIANA outperforms the other approaches. This effect becomes more prominent when an increasing numbers of jobs start arriving and queuing, and scheduling operations become challenging (due to huge data transfers and computing resource requirements).

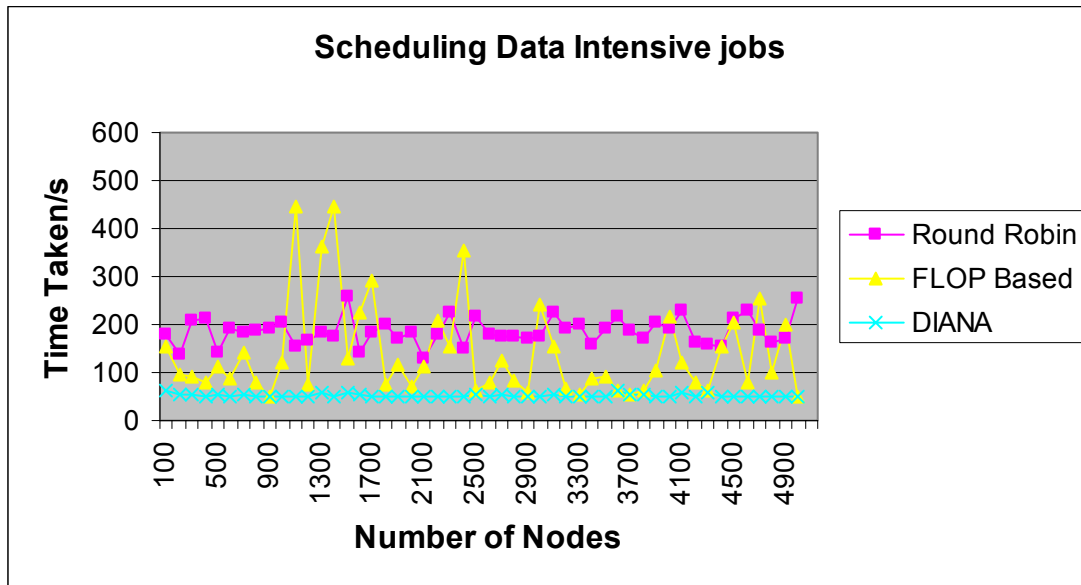


Figure 8.25: Comparison of DIANA for data intensive scheduling

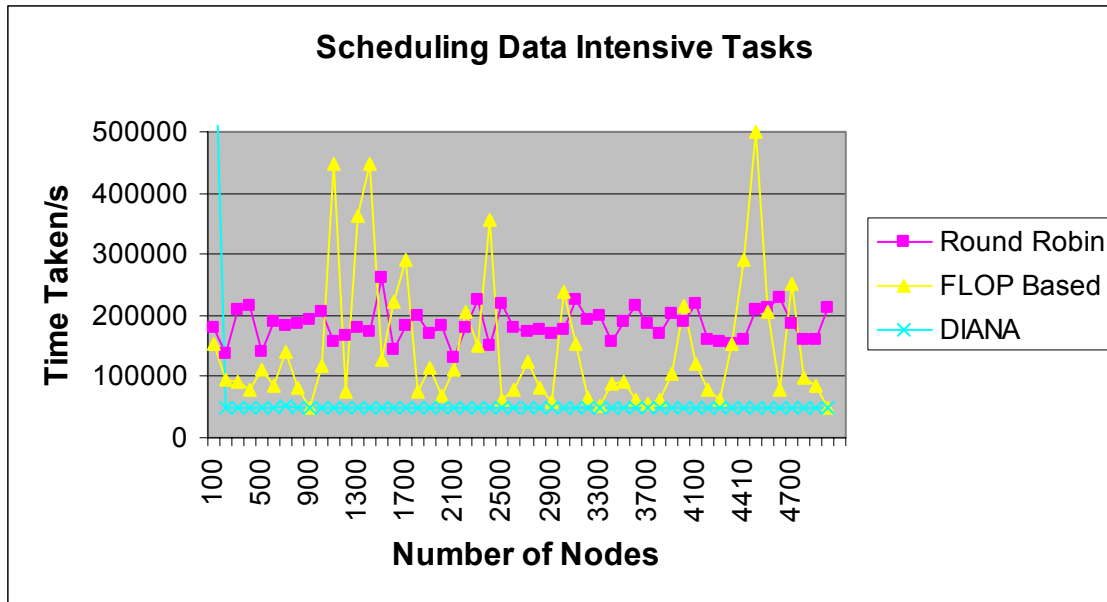


Figure 8.26: DIANA efficiency for data intensive jobs

After this, simulation jobs having the highest computation complexity (with higher data traffic between the jobs) were scheduled and the response noted in figure 8.26. In this particular test, jobs of 1GB were scheduled on the Grid and DIANA had stable and consistent performance, whereas Round Robin and FLOP showed a lot of variation as illustrated in figure 8.26. Some FLOP data points are invalid, that is why some points are nearly zero. As we can see in most figures of section 4.1, DIANA has the best performance in a wide number of scheduling scenarios, in efficiently scheduling compute and data intensive jobs preserving scalability and network efficiency.

The usage environment of DIANA is different to that of the two previous algorithms. DIANA is designed for P2P environments and not for centralized environments. Round Robin and FLOP are actually useful in centralized environments whereas DIANA cannot efficiently perform in such environments, because each node directly communicates with another node in DIANA scheduling. When it comes to centralized environments, there might be many tiers of sub-central environments in the Grid for scalability. For example modern Grids can be divided into Virtual Organizations, Clusters, etc. In such environments DIANA would have to go through the centralized hierarchy, finding the network connectivity with central servers before getting to the node at the site level which actually will process the job

or submit it. Thus any environment which uses DIANA ideally must be P2P, and sites must have the capability to directly connect to any other site on the Grid network.

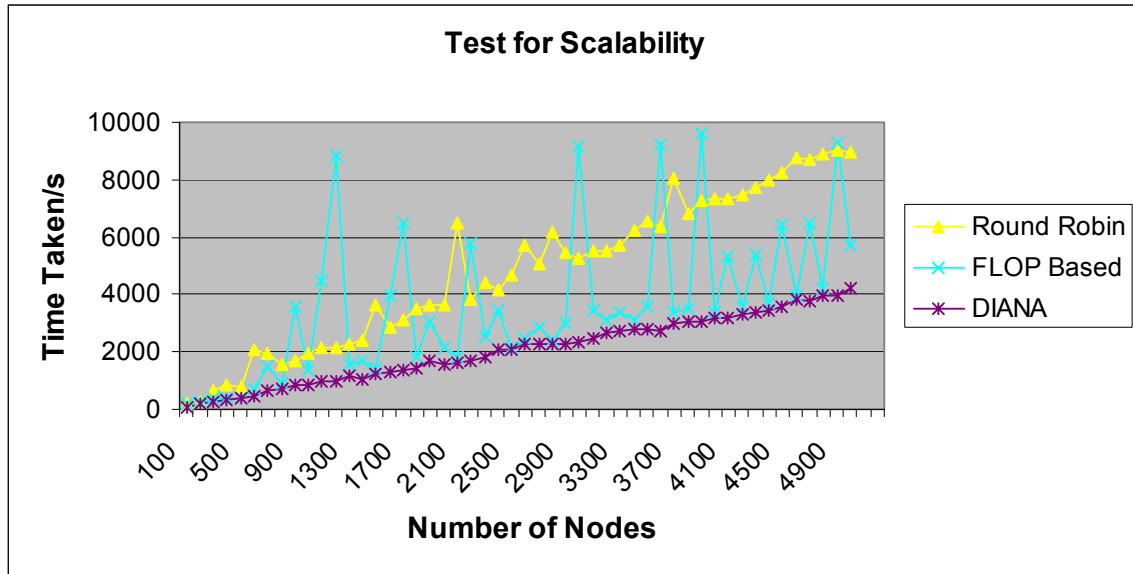


Figure 8.27: Scalability Test for DIANA

In conclusion we present here the results of the scalability tests for the DIANA scheduling approach. These are simulation results since it was not feasible to deploy the complete DIANA system on such a large number of sites. In these tests, we assumed that there is a meta-scheduler on each node (here, a node corresponds to a site), and all the nodes work in a P2P manner. As shown in figure 8.27, the number of nodes/sites and the number of jobs scheduled to the Grid was increased gradually to test which algorithm gives the steepest increase in time taken. An exponential increase reveals "poor" behaviour and shows that the algorithm is not scalable. In this test, jobs of a processing requirement of 3 MFLOP and a bandwidth load of 1 MB were launched to the Grid. The Round Robin Scheduler algorithm has a steep linear curve showing that it is the most non-scalable of the candidates. FLOP shows too much variation in this case, although it clearly is more scalable than Round Robin. The DIANA P2P approach has the best performance; it shows a nearly linear increase, and hence it is very scalable. This also indicates that DIANA is a suitable approach for large scale Grids since it can also support increasing numbers of Grid nodes. In an actual Grid deployment, there can not be any interference from other schedulers since there is either only one scheduler at each site or different schedulers at each site manage their own resources. Therefore, no two schedulers can interfere. Particularly the resources and schedulers are

dedicated in LCG, OSG, Nordu-Grid and EGEE testbeds, being used by CERN and its collaborators, and this makes the interference chances further minimal. This is further ensured in the DIANA Scheduling where all site schedulers work towards a common scheduling goal and inform and facilitate each other through information sharing and fault tolerance.

## **8.5 Conclusions**

In this chapter results of the scheduling process are presented using three different techniques. First a set of tests were performed as a result of the DIANA deployment on the GILDA testbed. It was demonstrated through these tests that data transfer, computation and network considerations could significantly optimize the data intensive scheduling process. It was also demonstrated that the network managed services and suitable selection of the network links between the sites before making the scheduling decisions was key to Grid optimization. Replica selection is crucial to data intensive scheduling; it depends on the network characteristics and an optimized replica selection leads to an optimized data intensive scheduling. These considerations not only improved the execution times of the jobs but also reduced the queue times of the jobs. It was identified that decentralized scheduling was not only suitable for data intensive bulk jobs, but it also improved the execution and scheduling process significantly. It was noted that the quality and accuracy of decisions in the P2P scheduling depended on the peer discovery and information propagation to other scheduling peers. It was demonstrated that in-memory cache is better suited for the rapid discovery of the scheduling instances and proved instrumental in realizing the improved performance in the DIANA scheduling. Job migration to better performing nodes can contribute further to the optimization of scheduling. It was also demonstrated that the multi-queue and priority-aware scheduling not only avoids starvation of the resources but also improves the quality of service. This priority and the multi-queue mechanism can significantly optimize the scheduling process, especially for the bulk scheduling and execution.

In conclusion, results related to bulk scheduling were presented. It was demonstrated that scheduling and execution can be optimized if jobs are migrated to those sites which have better execution capacity, have less load and queues and where the required data is also already available or can be made easily available. Furthermore, migration will enable lower

priority jobs to get high priority and their total execution times can be further reduced. Similarly, it was shown that this priority control can ensure the smooth allocation of resources so that no user can be subject to starvation by submitting millions of jobs at a time but rather jobs and users can be moderated by controlling the priority and migration procedure. It was also demonstrated that overall queue and execution time can be significantly reduced if the jobs having similar nature are co-scheduled in bulk. Furthermore, it was concluded that DIANA is better in scalability and consistency when compared to other contemporary scheduling approaches.

## Chapter 9

### Summary and Future Directions

This chapter is divided into three sections: Section 9.1 summarizes the whole thesis and presents a brief review of the issues and approaches which have been taken throughout the thesis text. Section 9.2 draws conclusions from the thesis findings and the findings are analysed and their significance is identified and discussed in this section. Section 9.3 discusses the future direction of the work which emerged during this research or which is related to those issues that were relevant but outside the scope of this research. Finally, the thesis is concluded with a closing statement.

#### 9.1 Summary

In this thesis we described a Data Intensive and Network Aware (DIANA) meta-scheduling approach which takes into account data, processing power and network characteristics when making scheduling decisions across multiple sites. The DIANA approach considers the Grid as a combination of active network elements and considers network characteristics as a first class resource in the scheduling decision matrix, along with computation and data. The scheduler can make “intelligent” decisions by taking into account the changing state of the network, the locality and the size of the data and the pool of processing cycles.

The research hypothesis investigated in this thesis asserted that:

Data intensive bulk scheduling can be significantly improved by taking into consideration a combination of network, data and compute costs, as well as by implementing effective queue management and priority control.

The research hypothesis has been proven through various experimental studies as discussed in chapter 8 and is supported by a series of theoretical, mathematical and architectural investigations in the previous chapters. Section 9.2 of this chapter will summarize the key findings and conclusions to support the hypothesis and associated research questions. The hypothesis was investigated and has been proven through experimentation and the research questions related to this hypothesis have been addressed in this thesis. Chapter 2 described the background of this thesis’s research and detailed the state-of-the-art in the data intensive

job scheduling domain. Related research was presented and an analysis of the current and previous efforts was discussed. Chapter 3 provided an analysis of the DIANA scheduling requirements and listed the salient features of such a system. Amongst those features it was established that data location is central to the working of an optimized Grid and hence it should be considered in scheduling decisions. It was also established that the ‘best’ or most optimal network path to computing and storage elements should be identified and that the scheduling system should calculate and incorporate network measurements while planning job submission. Chapter 3 further asserted that the network, computation and data transfer costs and the priority of job placement are vital when dealing with a large frequency of data intensive jobs and optimizing the scheduling decisions. A use case based study was discussed to identify and analyze the requirements for data intensive and network aware scheduling and these then led the research described in the later chapters of this thesis.

In chapter 4 a description of the DIANA scheduling algorithm and its key scheduling parameters was provided and how they influence optimization was discussed. It was shown, with the help of mathematical equations, that a candidate matrix of compute, network and data transfer costs can significantly optimize the scheduling process if each job is submitted and executed after taking into consideration certain associated costs. Queue time and site load, processing time, data transfer time, executable transfer time and results transfer time are the key elements which need to be optimized for optimization of scheduling and these elements were represented in the DIANA scheduling algorithm. The DIANA algorithm was extended and it was demonstrated that if queue, priority and job migration were included in the DIANA scheduling algorithm, the same algorithm could be used for the scheduling of bulk jobs. As a result, a multi-queue, priority-driven, feedback-based bulk scheduling algorithm which extends the DIANA scheduling algorithm was presented and illustrated.

Chapter 5 discussed the architecture of the Grid schedulers and related issues that can optimize the scheduling and execution process. Various scheduling hierarchies were discussed and it was concluded that centralized scheduling models are not adequate for complex scheduling scenarios, as in the case of bulk scheduling. We discussed how a P2P scheduling model would be ideal for the DIANA Scheduling and illustrated its queue management mechanism. We then described the DIANA scheduling architecture and illustrated the DIANA scheduling API. Chapter 6 described the process of extracting and

analyzing the network values and their impact on scheduling decisions. Network aware scheduling was discussed and its influence on scheduling decisions was described. This chapter also described the design and implementation of a Discovery Service and discussed its unique features such as flexibility, avoiding stale service data by imposing a lifetime for each service entry and providing an updated view of the dynamically varying Grid system.

In chapter 7, the role of the Data Location Service and data aware scheduling was explained. Chapter 7 also shed light on the replication and co-allocation issues for data intensive job scheduling and described the ways through which this could be managed in the DIANA Scheduler. It was illustrated that the scheduling efficiency could be improved by selecting the best replica of a dataset having the minimum access and transfer costs. It was illustrated that the data intensive scheduling process can be optimized by taking the replica locations and sorting these replicas based on network efficiency. The dataset replica having the least access cost with reference to the selected computing element would be selected by the Meta-scheduler.

## **9.2 Critical Analysis and Conclusions**

In Chapter 8 the results of tests related to the thesis research were discussed. Through experimental, graphical and analytical details it was demonstrated that DIANA Scheduling significantly optimizes the job queue and execution times. It was further demonstrated that bulk job scheduling and execution operations are improved by the decentralized, migration oriented, multi-queue and priority aware approach followed in the DIANA Scheduling. It was also concluded that a combination of data transfer cost, network cost and computation cost can significantly optimize the Grid scheduling and execution process which was the key message of the DIANA scheduling approach. These results validate the research questions and the hypothesis that data intensive bulk scheduling can be significantly improved by considering a combination of network, data and compute costs and by implementing effective queue management and priority controls.

The main conclusions of this research are the following:

- It was concluded that a comparative consideration of appropriate replicas and their computation and network costs can help in significantly optimizing data intensive job scheduling. It was concluded that it might be more logical and optimal to move the

data towards jobs in data intensive scheduling and some times both data and job movement to a third location can be helpful in scheduling optimization. It was also concluded that a considerable scheduling and execution optimization can be achieved if the network is treated as a resource and combination of the data transfer cost, network cost and computation cost is included in the scheduling decisions and this is the key message of the DIANA scheduling approach. *This addresses research question 1 as outlined in chapter 1 which asserts that co-allocation and co-scheduling is a key phenomenon for the scheduling optimization. This also validates research question 4 which emphasises the inclusion of data, compute and network costs in the optimization of data intensive scheduling.*

- This thesis demonstrated that network characteristics and the selection of stable connecting links optimizes the data intensive scheduling process. It was further demonstrated that replica selection depends on the network characteristics and suitable replica selection is essential to optimize data intensive job scheduling. These considerations not only improved the execution times of the jobs but also reduced the queue times of the jobs. *This validates research question 2 which states that the network aware services are required to optimize the data intensive scheduling.*
- It was demonstrated that decentralized scheduling significantly improves the execution and scheduling process. The level of the optimization cannot be exactly quantified due to the changing state of the resources having different capability at different sites but the approach roughly optimizes the execution and queue times around 20 % to 50 % on average. Also the quality and accuracy of the decisions in P2P scheduling depends on the peer discovery and information propagation to other scheduling instances. *This addresses research question 5 which establishes that the decentralized scheduling hierarchies are an effective approach for the data intensive bulk scheduling optimization.*
- It was further shown that multi-queue and priority aware scheduling not only avoids starvation of the resources but also improves quality of service (QOS) by ensuring the smooth allocation of resources. This priority and multi-queue mechanism can significantly optimize the scheduling process, especially when bulk jobs are being scheduled and executed. It was shown that job migration to better performing sites

can contribute to optimizing job scheduling. It was demonstrated that scheduling and execution can be optimized if jobs are migrated to those sites which have better execution capacity, less load, and where queues and required data are also available or can be made easily available. It was also demonstrated that overall queue and execution time can be significantly reduced if the jobs having similar nature are co-scheduled in bulk scheduling process. *This partially addresses research question 3 which suggests finding new techniques for the bulk scheduling optimization and then linking the priority and queuing with these techniques for overall scheduling optimization.*

- It was concluded that bulk jobs cannot be optimally scheduled by contemporary scheduling techniques. Converting the whole bulk into subgroups and then taking each subgroup as a single job, at the scheduling level, can improve the scheduling and execution process. This technique also enables one to use the existing scheduling system for this type of job which can be scheduled using the DIANA scheduling mechanism. *This addresses research question 3.*
- It was shown that the DIANA P2P approach is scalable at the level of thousands of nodes and performs much better than contemporary scheduling approaches. The DIANA P2P approach has the best performance; it shows a nearly linear increase, and thus is very scalable. This also demonstrates that DIANA is a suitable approach for large-scale Grids and can support increasing numbers of Grid nodes. *This in part addresses research question 5 which outlined the scalability concerns in the DIANA scheduling approach.*
- It was also demonstrated that DIANA is not only scalable but at the same time it is also a very stable scheduling approach. It does not respond exponentially and does not face congestion as the number of jobs increases beyond a certain capacity or the number of nodes increases beyond a certain limit. Therefore it is a rather stable approach for a large number of jobs and nodes and is suitable for large-scale Grid projects. *This resolves the concerns outlined in research question 5 about the stable nature of the DIANA approach for the bulk scheduling optimization.*

These conclusions endorse the hypothesis described in chapter 1. These conclusions also adequately answer the research questions stated in chapter 1. This work is unique in the

sense that it tried to resolve a real life problem being faced by the LHC analysis community but at the same time it can be applied to other areas of Grid scheduling especially those domains where a lot of the data, computation and long distance data transfer calls are involved. Furthermore it can handle “chaotic” job scenarios and can ensure quality of service even in the worse case scenarios such as High Energy Physics analysis where thousands of users submit millions of jobs to analyze the data distributed and replicated all over the world.

Due to the emerging scientific and business applications, more and more data is being stored and needs to be analysed. Scheduling computing resources for the analysis of this data can be done quickly and effectively by using the issues and concerns discussed in this thesis. There are a number of other fields beyond the Physics analysis domain where these results can be applied, for example, earth sciences, bioinformatics, neurosciences, drug discovery, climate modelling, aerospace science and weather forecasting, to name but a few. Even the major software industry giants like Cisco, HP, Oracle, IBM and Microsoft are working in the similar direction to launch their products, for example Noemix, Parabon, Sun Grid Engine, Platform LSF, PBS, Avaki, Legion, Data Synapse, Entropia, and this thesis research could be very beneficial for these and other products and companies.

Four major areas in computer science can benefit from this research. Distributed and Grid computing is the major beneficiary since it can now also provide solutions to very complex problems which require distributed wisdom; the use and economy of resources can be managed which otherwise can be difficult to achieve. More precisely, the resource management and scheduling in distributed computing has been advanced by providing a next generation self-organizing solution which can deal with almost any scheduling eventuality. The concepts and solutions discussed in this thesis can equally be extended and applied to operating systems, the second area of computer science to benefit from this research. Work has already been started towards the next generation Grid Operating System (GridOS) to make the Grid a ubiquitous problem solving environment which can be accessed and used even by everyday users. We have published a concept paper [50] in which we emphasized that most of these concepts and solutions will become part of a Grid operating system since all operating systems in essence do some sort of resource management. Due to the nature and complexity of Grid applications, it is most feasible to port the Grid services to the operating system thus enabling plug and play, self-discoverable Grid computing.

The third major area in computer science that benefits from this research is the Grid middleware itself. Up to now, most of the services and solutions have been advocated by assuming that the underlying network is intelligent and therefore it will automatically take care of the application needs. Unfortunately, this is not the case in reality and by introducing the concept of network managed services and application aware networks, we have laid the foundation for the next generation middleware which should be coupled with the network characteristics and should respond according to the network and application needs. It will be difficult for existing middleware to support the next generation of Grid applications due to the reasons stated earlier and this research has paved the way for new middleware which should not only meet the needs of emerging applications but should also can ensure a high level of optimization. Moreover, as established in previous chapters the centralized Grid middleware solutions are not sufficient for emerging compute and data-intensive problems and a decentralized architecture is more effective, suggesting that the middleware domain should move to a self managing, scalable and fault tolerant middleware in general and Grid resource management systems in particular.

The fourth area of computer science application is the mining and analysis of the data distributed in remote locations. We have observed an explosive growth in the number and size of data being produced around the world from the scientific and business applications in the last few years. This gave researchers the opportunity to develop effective data mining techniques for discovering and extracting information from huge amounts of data. But due to their size and also to social or legal restrictions that may prevent analysts from gathering data in a single site, the datasets are often physically distributed. The analysis and mining process is data and computationally expensive and the Grid is a natural platform for deploying a high performance service for the parallel and distributed information discovery process. The Grid environment may in fact furnish coordinated resource sharing, collaborative processing, and high performance data mining analysis of the huge amounts of data produced and stored. Since these applications are typically data intensive, one of the main requirements of such a Grid environment is the efficient management of storage, computation and communication resources and this is the area where the DIANA scheduling technique can be very effective in optimizing the analysis process. DIANA can enable the distributed database community to quickly analyze large datasets and optimize their algorithms and applications by efficiently exploiting the computing and storage resources.

There are certain limitations which should be considered when applying the results of this research. We have adopted simulation results to demonstrate the DIANA bulk scheduling process. We could have deployed the DIANA bulk scheduling on the EGEE, OSG, NORDU etc Testbeds and ideally should have tested it on different transatlantic Testbeds to investigate the effect, limitation and other problems which can be faced by such a scheduling system when coping with the job frequency from hundreds of users and the job scheduling on resources distributed all over the world. Due to the nature of the jobs spanning days and consuming huge amounts of data, this was not possible on current Grid deployments. The simulation results that were used for bulk scheduling were as close to the real world environment as was possible however the behaviour of the actual Grid deployment might eventually vary due to the nature of the jobs and scheduling requests from different users and administrative limitations and policies. For example, some users may want to run all of their jobs on a single site or there might be the case that the VO admin does not allow the jobs to run on some site beyond a certain limit and in this case, the scheduler cannot take the optimized decision due to these limitations. Therefore this can be the biggest limitation to generalizing and applying the results of this thesis research to all Testbeds.

Grid deployments should follow the policies and requirements outlined in this thesis to realise the true benefit of the optimization. In other words, VO policies can have strong influences on the level of the scheduling optimization which can be achieved by deploying the DIANA scheduling approach. Furthermore, decentralized scheduling provides better availability, fault tolerance and effective bulk scheduling through job migration, but produces more network traffic, is less flexible, and is consequently more difficult to implement. The steering and migration of jobs is optimal only for long running jobs and the scheduler should not migrate short running jobs due to the severe associated penalties. There are many ways to determine the real bandwidth between two nodes, however there are limitations to each, we base our bandwidth determination mechanism on TTL of ICMP packets [40]. Many organizations block ICMP traffic, thus our bandwidth determination method would not work in those environments. Other methods such as, using SNMP to determine bandwidth, are not feasible over different networks, as many routers are not yet SNMP enabled, and organizations disable SNMP support from their routers for security purposes. Besides these two techniques there are a number of bandwidth intrusive techniques, such as Iperf which have their own inherent problems. The accuracy and quality of the network information can

enhance the quality and accuracy of the scheduling decisions. Moreover, the stability of the network links is crucial for quality scheduling since otherwise assumptions and predictions made at a particular time might not be valid at a later point in time and therefore can lead to non-optimal scheduling. These and other related limitations such as changing network characteristics due to traffic and congestion problems, the availability of the resources and the site and job failures on Grid Testbeds, may limit the application of these thesis results.

### **9.3 Future Direction**

There are many other related aspects of the research area beyond those issues stated above which are important in data intensive bulk scheduling research but were not covered in the DIANA scheduling approach. For example, the failure and re-submission of jobs is not covered and further research is needed for future fault tolerant job scheduling. The Scheduler should be aware of failing jobs and such jobs should be resubmitted to stable sites in the case of a failure. Furthermore, when jobs are failing on a site it should be evaluated whether the jobs should continue to be sent to that site or whether the scheduler should select a new more appropriate site based on some site reliability index. Moreover, the Scheduler should also be aware of the execution environment on a particular node prior to job submission and methods need to be established for providing this functionality. Data persistency and permanent storage is another issue which needs to be investigated. The scheduler should also have the capability to decide whether to store or cache the data on a particular location for possible later use. Potential Schedulers should also be parameterized so that certain jobs would always be scheduled by a particular scheduler tuned for that type of jobs thereby increasing the job scheduling and execution efficiency. It also needs to be determined whether parameterized schedulers can contribute to meta-scheduling optimization. These are just some of the research issues which were not able to be covered in this thesis but merit future investigation..

#### **9.3.1 Potential Limitations of the DIANA Algorithm**

DIANA mainly relies on performance information collected in the past in order to schedule jobs, i.e. to “predict” the future. Therefore, the system relies on the fact that the future is assumed to be similar to the past and this is a potential problem in all forecasting systems. In addition, since the cost model depends on very detailed and up-to-date monitoring

information, DIANA relies very heavily on the stability, scalability and accuracy of the underlying monitoring system. Like many meta-schedulers DIANA uses the push approach where jobs are actually pushed (or sent) to computing elements. This approach has the disadvantage that jobs might fail due to configuration errors at the destination sites. Schedulers with pull approaches can take care of this problem. Another way to overcome this is to send jobs that constantly monitor the environment for potential errors. All these issues need to be considered in future research surrounding data intensive job scheduling.

### **9.3.2 The Comparison of Bulk Scheduling Algorithms**

The comparative performance of the bulk scheduling algorithm discussed in chapter 4 needs to be evaluated. There is no competitive algorithm available against which we can compare the performance and effectiveness of the discussed bulk scheduling algorithm since it is the initial work of its kind, especially splitting the jobs into groups and subgroups is quite novel. Therefore we could not compare with existing approaches or create some alternative approach to our bulk scheduling approach. There is the possibility of customizing available algorithms like the gang scheduling algorithm that is an algorithm for parallel jobs and provides strong inter-process communication. Endeavour should be made to test and compare our bulk scheduling algorithm with different variants of gang scheduling and other contemporary algorithms. One possible approach would be to customize the gang scheduling algorithm for non-parallel environments, align it to the bulk scheduling needs and then compare it with the bulk scheduling algorithm we discussed in chapter 4.

### **9.3.3 Network Managed Scheduling**

Load balancing and scheduling has traditionally been carried out at the application level but due to the changing nature of applications, *application-aware* networks and *network-aware* applications are becoming increasingly recognised as pre-requisites for true end-to-end scheduling optimization. While network routing ensures that packets take the ‘best’ source to destination route, it should be the application that decides the communication end points. Being able to choose the communication end points requires that the application has access to the network information and the server load. It is therefore essential to have the services that understand the dynamics of the underlying network infrastructure at the packet level. This will enable scheduling applications to communicate by routing application messages to the

appropriate destination, in the format expected by that destination. This will allow the scheduler's message-level intelligence into the network to better meet the underlying needs of applications for real-time visibility, security, event-based messaging, optimized delivery, and other core integration and deployment services. As a consequence future scheduling systems need to be created which can autonomously estimate the network qualities between the end points including bandwidth, topology and load and can then autonomously balance the Grid work load. Such a scheduling system should self-manage, self-organize and self-heal themselves in case of failures.

### **9.3.4 From Grid Middleware to a Grid Operating System**

Existing Grid middleware supports only primitive forms of resource management. Grid middleware such as Globus mostly facilitates Grids which are aggregations of clusters. Various schemes based on usage policies, resource reservation and quota and accounting have been implemented, however these do not provide effective resource management in which collections of clusters can look like a single virtual machine. Most of these issues can be resolved effectively by migrating many of the functions that Grid middleware has traditionally been responsible for to the operating system level. Migrating Grid computing functions to the operating system level would make it transparent and invisible to the user and would thereby remove the obstacles related to the installation, management and maintenance of Grid middleware. Grid enabling process management in the operating system would yield a completely new paradigm for developing Grid enabled applications, which could remove many of the barriers related to the re-engineering of applications for execution on Grids, and could provide universal support for many more types of applications including support for legacy desktop applications.

Resource management which has been carried out traditionally at the application level in Grid middleware could be done more effectively at the operating system level, since at their heart (distributed) operating systems are basically resource management systems. Migrating Grid computing to the operating system, would remove any differences between non-Grid computing systems and Grid computing enabled systems, thus making Grid computing ubiquitous. We envisage a brokering and scheduling engine inside the operating system

kernel which takes into consideration entire pools of resources available across the Grid. Further details are available in the paper [50].

### **9.3.5 Pre-emptive Scheduling of the Bulk Jobs**

We asserted throughout this thesis that we are using a non pre-emptive job execution model due to the costs involved in the data transfer and other limitations of the Grid environment. However there are some cases where checkpointing and state migration of data intensive jobs might be useful and therefore a pre-emptive mode of DIANA scheduling ought to be considered. This might lead to the scheduling optimization although almost all major Grid projects are currently considering only the non pre-emptive approach. Applications might eventually emerge which can produce optimized results if they are executed in a pre-emptive scheduling mode. The steering and migration of such jobs would be a challenging task but should be considered in future research.

Furthermore, bulk jobs are executed in groups which might take days to complete and due to failures of the machine or increases in load this process may even take weeks. Consequently, it will be cost effective to migrate and export the whole bulk submission to some other site which may produce results very much more quickly than the current site. Currently this is not possible until we can also migrate the previous execution state of the jobs since at present the execution time is likely to lengthen. All these aspects need due consideration in future research.

### **9.3.6 Meta-Scheduling Peers Grouping**

In DIANA, every site meta-scheduler communicates with its peers on other sites. This may become problematic when the size of the Grid grows beyond a certain limit due to communication overhead involved since each meta-scheduler will propagate its information to other peers at all the Grid sites and this may choke the network. There should be a region wise grouping of the peers so that sites in each region might first talk to each other for effective scheduling and should contact other peers beyond that region if the jobs requirements cannot be met within that group of sites. This will restrict the traffic within the group and only limited traffic will travel across the whole Grid. This will not only restrict the traffic size but will also reduce the amount of the data replication across the long distance

sites and will facilitate the job migration process. There can be further investigation to consider a group meta-scheduler if the Grid size becomes too big.

### **9.3.7 Workflow Optimization through DIANA**

In this thesis the jobs undertaken for scheduling, for example the physics jobs, did not have mutual dependencies. The DIANA approach could be extended in future to actually optimize the workflow of the dependent jobs. This optimized workflow would be constructed following the DIANA characteristics and would become an input to the scheduler for further optimizing the overall scheduling and execution process. There are a number of applications where jobs are tightly coupled and depend on each other, for example the bioinformatics jobs, and this step would enable DIANA to emerge as a ubiquitous scheduling environment.

## **9.4 Closing Statement**

Grid computing is poised to become the platform to support emerging applications which need huge resources and are unable to be managed by the existing clustering or supercomputing techniques. To enable the Grid to efficiently employ the distributed resources, an efficient scheduling system is required which can make informed decisions by taking into account the state of the distributed resources. In this thesis, the DIANA Scheduling approach was presented which can significantly optimize the data intensive scheduling process. It can support the huge frequency of the jobs on the Grid to access and analyse this data by co-allocating and co-scheduling the data and computation power. Network aware scheduling decisions, regulated by the priority driven algorithms working in decentralized mode, creating multi-queues and converting and scheduling the bulk jobs into the homogeneous clusters and groups can significantly optimize the queuing and execution times.

## Bibliography

- [1] I. Foster and C. Kesselman (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.
- [2] M. Baker, R. Buyya, D. Laforenza, The Grid: International Efforts in Global Computing, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000), l'Aquila, Rome, Italy, July 31 - August 6. 2000.
- [3] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A Resource Management Architecture for Metacomputing Systems, 4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, Florida March 30, 1998.
- [4] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, International Conference on High Performance Computing in Asia- Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, 2000.
- [5] J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds), Grid Resource Management. Kluwer Publishing, Fall 2003.
- [6] P. Andreetto, S. Borgia, A. Dorigo, A. Gianelle, M. Mordacchini et. al, Practical Approaches to Grid Workload & Resource Management in the EGEE Project, CHEP04, Interlaken, Switzerland.
- [7] Sun Grid Engine, <http://www.sun.com/software/Gridware/>
- [8] J. Basney, M. Livny, and P. Mazzanti, Utilizing Widely Distributed Computational Resources Efficiently with Execution Domains. Computer Physics Communications, 2001.
- [9] J. Brooke, D. Fellows and J. MacLaren, Resource Brokering: The EUROGRID/GRIP Approach, Proceedings of the UK e-Science All Hands Meeting 2004, 31st August - 3rd September, Nottingham UK
- [10] gLite Computing Element, <http://glite.web.cern.ch/glite/ce/>
- [11] I. Legrand, MonaLisa - Monitoring Agents using a Large Integrated Service Architecture, International Workshop on Advanced Computing and Analysis Techniques in Physics Research, Tsukuba, Japan, December 2003
- [12] L. Cottrell, W. Matthews, Measuring the Digital Divide with PingER, Second round table on Developing countries access to scientific knowledge, pp 23 – 24, Trieste, Italy, October 2003.
- [13] P. Steenkiste, Adaptation Models for Network-Aware Distributed Computations, 3rd Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, Orlando, January, 1999.

- [14] Tierney, B. Lee, J., Crowley, B., Holding, M., Hylton, J., Drake, F., A Network-Aware Distributed Storage Cache for Data Intensive Environments, Proceeding of IEEE High Performance Distributed Computing conference (HPDC-8), August 1999, LBNL-42896.
- [15] GILDA (Grid Infn Laboratory for Dissemination Activities <https://gilda.ct.infn.it/>
- [16] GGF Grid Scheduling Architecture group, <https://forge.Gridforum.org/projects/gsa-rg>
- [17] Kavitha Ranganathan ,Jan Foster, Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications , HPDC 2002.
- [18] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauv'e, F. A. B. da Silva, C. O.Barros, and C. Silveira, Running bag-of-tasks applications on computational Grids: The myGrid approach, in Proceedings of the ICCP'2003 – International Conference on Parallel Processing, October 2003.
- [19] J. Smith and S. K. Shrivastava, A system for fault tolerant execution of data and compute intensive programs over a network of workstations, in Lecture Notes in Computer Science, vol. 1123, IEEE Press, 1996.
- [20] Sebastian Ho, Survey of Grid Meta-Scheduler,  
[http://student.bii.astar.edu.sg/~sebastianh/Scheduler\\_Survey.pdf](http://student.bii.astar.edu.sg/~sebastianh/Scheduler_Survey.pdf), Oct.2002.
- [21] Baru, C., Moore, R., Rajasekar, A., and Wan, M. The SDSC storage resource broker. In Proceedings of the 8th Annual IBM Centers for Advanced Studies Conference (Toronto, Canada, 1998).
- [22] Eddy Caron, Vincent Garonne, Andreï Tsaregorodtsev, Evaluation of Meta-scheduler Architectures and Task Assignment Policies for High Throughput Computing, Research Report No 2005-27, Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon
- [23] Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics for Scheduling Jobs on Computational Grids. International Conference on Advanced Computing and Communications (2000)
- [24] P. Avery and I. Foster, The GriPhyN Project: Towards Petascale Virtual Data Grids," Technical Report GriPhyN-2001-15, 2001.
- [25] M. Neary, P. Cappello, Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing. Joint ACM Java Grande - ISCOPE Conference (2002)
- [26] Abramson, R Buyya , J. Giddy, A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems Journal, Volume 18, Issue 8, Elsevier Science (2002) 1061-1074

- [27] G. Allen, D. Angulo, I. Foster et. al , The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *Journal of High-Performance Computing Applications*, Volume 15, no. 4 (2001)
- [28] E. Huedo, Rs. Montero, Llorente, An Experimental Framework for Executing Applications in Dynamic Grid Environments. *ICASE Technical Report* (2002).
- [29] S.S Vadhiyar, J. Dongarra, A Performance Oriented Migration Framework for the Grid. <http://www.netlib.org/utk/people/JackDongarra/papers.htm> (2002)
- [30] W. Bell, D. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Design of a Replica Optimisation Framework. Technical Report, DataGrid-02-TED-021215, Geneva, Switzerland, December 2002.
- [31] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lemaitre, G. McCance, H. Stockinger, K. Stockinger, et al., Replica Management in the EU DataGrid Project, *International Journal of Grid Computing*, 2(4):341-351, 2004.
- [32] K. Stockinger, H. Stockinger et al. Access Cost Estimation for Unified Grid Storage Systems. In 4th International Workshop on Grid Computing (Grid2003), Phoenix, Arizona, November 17, 2003. IEEE Computer Society Press.
- [33] Y. Zhao, Y. Hu, GRESS - a Grid Replica Selection Service, ISCA 16th International Conference on Parallel and Distributed Computing Systems (PDCS-2003), 2003.
- [34] C. Tan, K. Mills, Performance Characterization of Decentralized Algorithms for Replica Selection in Distributed Object Systems, *Proceedings of the 5th International Workshop on Software and Performance (WOSP'05)*, Palma de Mallorca, Spain, July 12-14, 2005.
- [35] J. Nabrzyski, Knowledge-based Scheduling Method for Globus. Globus Retreat, Redondo, 1999. <http://www.man.poznan.pl/metacomputing/ai-meta/globusnew/index.html>.
- [36] A. Chervenak et al., Giggie: a framework for constructing scalable replica location services', *Proceedings of Supercomputing 2002 (SC2002)*, Baltimore Maryland, November 16-22, 2002.
- [37] T. Barras et al., The CMS PhEDEx system: a novel approach to robust Grid data management, UK All Hands Meeting, 2005.
- [38] T. Kosar and M. Livny, A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems, To appear in *Journal of Parallel and Distributed Computing*, 2005.
- [39] D. Thain et al., Gathering at the well: creating communities for Grid I/O, *Proceedings of Supercomputing 2001*, November, Denver, Colorado.
- [40] J. Postel, Internet control Message Protocol (ICMP), Network Working Group, Request for Comments:792, ISI 1981 <http://www.faqs.org/rfcs/rfc792.html>

- [41] Elizeu Santos-Neto, Walfredo Cirne, Francisco Brasileiro, Aliandro Lima, Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids, Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing - June 2004.
- [42] S. Park, and J. Kim, Chameleon: a resource scheduler in a data Grid environment, Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, Tokyo, 2003
- [43] Grid Community Scheduling Framework, <http://sourceforge.net/projects/gcsf/>
- [44] Lauret et al., The STAR Unified Meta-Scheduler project, a front end around evolving technologies for user analysis and data production.CHEP2004,Interlaken Switzerland.
- [45] Henri Casanova, Arnaud Legrand and Loris Marchal, Scheduling Distributed Applications: the SimGrid Simulation Framework, Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03).
- [46] J.Uk et al., SPHINX: A Scheduling Middleware for Data Intensive Applications on a Grid, CHEP 2004, Interlaken Switzerland, 2004.
- [47] H.Daily et al., A Decoupled Scheduling Approach for the GrADS Program Development Environment, Supercomputing 2002, Baltimore, Mayland, November 16-22, 2002 and Application Level Scheduling (AppLeS) <http://apples.ucsd.edu/>.
- [48] Elmroth, E. and Gardfjall, P. 2005. Design and Evaluation of a Decentralized System for Grid-wide Fairshare Scheduling. In First international Conference on E-Science and Grid Computing 05
- [49] H. Jin, X. Shi et al. An adaptive Meta-Scheduler for data-intensive applications, International Journal of Grid and Utility Computing 2005 - Vol. 1, No.1 pp. 32 - 37
- [50] Richard McClatchey, Ashiq Anjum et al., From Grid Middleware to a Grid Operating System Arshad Ali. The 5th International Conference on Grid and Cooperative Computing (GCC 2006) Changsha, Hunan, China 2006.
- [51] L. Boloni and D. Marinescu, An Object-Oriented Framework for Building Collaborative Network Agents, in Agents in Intelligent Systems and Interfaces, Kluwer Publishing House 1999.
- [52] P. Chandra, A. Fisher, C. Kosak et al., Darwin: Customizable Resource Management for Value-Added Network Services. 6th IEEE International Conference on Network Protocols, 1998.
- [53] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A Directory Service for Configuring High-Performance Distributed Computations, 6th IEEE Symp. on High-Performance Distributed Computing, 1997.

- [54] M. Neary, A. Phipps, S. Richman, P. Cappello, Javelin 2.0: Java-Based Parallel Computing on the Internet, Proceedings of European Parallel Computing Conference (Euro-Par 2000), Germany, 2000.
- [55] G. Coulson and M. Clarke, A Distributed Object Platform Infrastructure for Multimedia Applications, Computer Communications Vol. 21, No. 9, July 1998, pp. 802-818.
- [56] J. Gehring and A. Streit, Robust Resource Management for Metacomputers, 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, USA, 2000.
- [57] D. Hensgen, T. Kidd, et al., An Overview of MSHN: The Management System for Heterogeneous Networks, 8th Workshop on Heterogeneous Computing Systems (HCW '99), San Juan, Puerto Rico, April 1999, invited.
- [58] H. Casanova and J. Dongarra, NetSolve: A Network Server for Solving Computational Science Problems, Intl. Journal of Supercomputing Applications and High Performance Computing, Vol. 11, Number 3, 1997.
- [59] N. Kapadia, R. Figueiredo, PUNCH: Web Portal for Running Tools, IEEE Micro, -June, 2000.
- [60] S. Gribble, M. Welsh, E. Brewer, D. Culler, The Multispace: an Evolutionary Platform for Infrastructural Services, Proceedings of Usenix Annual Technical Conference, Monterey, CA, June 1999.
- [61] Mark Silberstein, Dan Geiger, Assaf Schuster, and Miron Livny, Scheduling Mixed Workloads in Multi-Grids: The Grid Execution Hierarchy, Proceedings of the 15th IEEE Symposium on High Performance Distributed Computing (HPDC), 2006.
- [62] S. Chapin, J. Karpovich, A. Grimshaw, The Legion Resource Management System, Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999.
- [63] I. Legrand, H. Newman et al., The MONARC Simulation Framework, Workshop on Advanced Computing and Analysis Techniques in Physics Research, Japan 2003
- [64] <http://www.clusterresources.com/products/maui/docs/8.2backfill.shtml>
- [65] Hubertus Franke, Mike Kravetz , Priority-Queue Scheduler for Linux, IBM Linux Technology Centre <http://lse.sourceforge.net/scheduling/PrioScheduler.html>
- [66] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita, Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications, Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), , pp. 34-43, 2003.06

- [67] R.Lewis, B. Paechter, An Empirical Analysis of the Grouping Genetic Algorithm: the Timetabling Case, The 2005 IEEE Congress on Evolutionary Computation pp. 2856-2863. Edinburgh, Scotland: IEEE. 0-7803-9363-5
- [68] Patrick Briest and Piotr Krysta, Single-Minded Unlimited Supply Pricing on Sparse Instances, In Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006.
- [69] Noriyuki Fujimoto and Kenichi Hagihara, A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks, SAINT 2004 Workshop on High Performance Grid Computing and Networking, IEEE Press, pp.674-680, Tokyo Japan, January 26-30, 2004
- [70] Eduardo Hudo, Ruben S. Montero, and Ignacio M. Llorente, The GridWay Framework for Adaptive Scheduling and Execution on Grids Scalable computing: practice and experience (SCPE,), 2005.
- [71] Internet Engineering Task force, [www.ietf.org](http://www.ietf.org)
- [72] Distributed Management Task force, <http://www.dmtf.org/home>
- [73] Organization for the Advancement of Structured Information Standards, [www.oasis-open.org/](http://www.oasis-open.org/)
- [74] World Wide Consortium, [www.w3c.org](http://www.w3c.org)
- [75] T.Coviello, et al., Bridging Network Monitoring and the Grid, CESNET Prague, 2006.
- [76] Web services resource framework,  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsrf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf)
- [77] Distributed Resource Management Application API,  
<http://www.ggf.org/documents/GWD-R/GFD-R.022.pdf>
- [78] Web Based enterprise Management, <http://www.dmtf.org/standards/wbem/>
- [79] Universal Plug and Play, [http://msdn.microsoft.com/library/en-s/wceupnps/html/\\_wcecomm\\_universal\\_plug\\_and\\_play\\_upnp.asp](http://msdn.microsoft.com/library/en-s/wceupnps/html/_wcecomm_universal_plug_and_play_upnp.asp)
- [80] Configuration Description, Deployment, and Lifecycle Management WG (CDDL-M-WG), <http://forge.Gridforum.org/projects/cddl-m-wg>
- [81] GGF Standard on Advance Reservation, <http://www.ggf.org/documents/GFD/GFD-E.5.pdf>
- [82] GGF Grid monitoring architecture, <http://www.ggf.org/documents/GFD/GFD-I.7.pdf>
- [83] The Simple Network Management Protocol,  
<http://www.cisco.com/warp/public/535/3.html>

- [84] Grid High Performance Networks, <https://forge.Gridforum.org/projects/ghpn-rg>
- [85] [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsdm](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm)
- [86] YarKhan, A., Dongarra, J., Experiments with Scheduling Using Simulated Annealing in a Grid Environment. Workshop on Grid Computing, Baltimore (2002)
- [87] Ian Foster, R. L. Grossman, Data integration in a bandwidth-rich world. Communications of ACM 46, 11 (Nov. 2003), 50-57. DOI=<http://doi.acm.org/10.1145/948383.948409>
- [88] Super Computing 2005 Demonstration  
<http://www-iepm.slac.stanford.edu/monitoring/bulk/sc2005/hiperf.html>
- [89] CMS experiment at CERN <http://cms.cern.ch/>
- [90] Moab Grid Suite <http://www.clusterresources.com/pages/products/moab-Grid-suite.php>
- [91] Vincenzo Di Martino, Marco Mililotti, Scheduling in a Grid Computing Environment Using Genetic Algorithms, ipdps, p. 0235, International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops, 2002.
- [92] Kento Aida, Wataru Natsume, Yoshiaki Futakata, Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm, CCGrid, p. 156, 3rd International Symposium on Cluster Computing and the Grid, 2003.
- [93] Xiaoshan He, Xianhe Sun, Gregor von Laszewski, QoS Guided Min-Min Heuristic for Grid Task Scheduling ,Journal of Computer Science and Technology 2003 Volume: 18 Issue: 4 p. 442 - 451
- [94] R. L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal, 45:1563–1581, 1966.
- [95] Rafael Moreno , Job Scheduling and Resource Management Techniques in Dynamic Grid Environments, The First European Across Grids Conference 2003, Universidad de Santiago de Compostela, Spain
- [96] Atkinson, M., Chervenak, A., Kunszt, P., Narang, I., Paton, N., Pearson,D., Shoshani, A., and Watson, P. Data access, integration, and management. In The Grid: Blueprint for a New Computing Infrastructure, 2nd Ed.,I. Foster and C. Kesselman, Eds. Morgan Kaufmann, San Francisco, CA, 2004.
- [97] Beck, M., Moore, T., and Plank, J. An end-to-end approach to globally scalable network storage In Proceedings of ACM Sigcomm'02 (Pittsburgh, PA, Aug. 19–23). ACM Press, 2002.
- [98] C. Altinbuken, L. Yao, and S. Subramaniam, A Comparison of Fair Queuing Algorithms for the 2003 Internet, Communications and Information Technology Scottsdale, AZ, USA

- [99] [http://en.wikipedia.org/wiki/Category:Scheduling\\_algorithms](http://en.wikipedia.org/wiki/Category:Scheduling_algorithms)
- [100] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In the 8th IEEE Heterogeneous Computing Workshop (HCW'99), pages 30–44, 1999.
- [101] Chris Johnson, Basic Research Skills in Computing Science, Glasgow Interactive Systems Group (GIST), Department of Computer Science, Glasgow University, Glasgow, G12 8QQ.
- [102] Barry Boehm, A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes, August 1986.
- [103] Barry Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer, vol.21, #5, May 1988, pp 61-72.
- [104] Craig Larman, Philippe Kruchten, Kurt Bittner, How to Fail with the Rational Unified Process: Seven Steps to Pain and Suffering, Valtech Technologies & Rational Software, 2001
- [105] Ian Foster, Grid's place in the service-oriented architecture, ComputerWorld November 29, 2004  
<http://www.computerworld.com/databasetopics/data/story/0,10801,97919,00.html>
- [106] Eitan Frachtenberg, Dror G. Feitelson, Flexible Co Scheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources, International Parallel and Distributed Processing Symposium (IPDPS'03), April 2003.
- [107] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational Grids. In Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), August 1999.
- [108] MySQL database <http://dev.mysql.com/>
- [109] CMS Production, <http://cmsdoc.cern.ch/cms/production/www/html/general/>
- [110] R. Fruhwirth, M. Regler, R K Bock, H Grote, D Notz, Data Analysis Techniques for High-Energy Physics, Cambridge University Press - ISBN: 0521635489, page 121.
- [111] Koen Holtman, CMS Data Grid System Overview and Requirements, Division of Physics, Mathematics and Astronomy, California Institute of Technology, CMS NOTE 2001/037, <http://kholtman.home.cern.ch/kholtman/cmsreqsweb/cmsreqs.html>
- [112] F. Carminati et al., LHC Grid Computing Project Common Use Cases For a HEP Common Application Layer Model, HEP CAL RTAG Report 2004. [http://project-lcg-gag.web.cern.ch/project-lcg-gag/LCG\\_GAG\\_Docs/HEPCAL-prime.doc](http://project-lcg-gag.web.cern.ch/project-lcg-gag/LCG_GAG_Docs/HEPCAL-prime.doc)

- [113] D. Angulo et al., Toward a Framework for Preparing and Executing Adaptive Grid Programs. IPDPS, 2002.
- [114] EGEE Data Scheduler <http://egee-jra1-dm.web.cern.ch/egee-jra1-dm/>
- [115] Alex Olugbile et al., New Grid Scheduling and Rescheduling Methods in the GrADS Project, The Workshop for Next Generation Software, Santa Fe, New Mexico, April 2004, held in conjunction with the IEEE International Parallel and Distributed Processing Symposium 2004.
- [116] A. Chervenak, I. Foster, C. Kesselman, et al, The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. J. Network and Computer Applications (2000)
- [117] Pat Schuler, Tom Shull, Product Requirements development and Management Procedure, NASA Program and Project Management Processes and Requirements.
- [118] Grossman R., Gu Y., Hanley, D., Hong, X., etc all., and Weinberger, J. Experimental studies using photonic data services at iGrid 2002. Future Gen. Comput. Syst. (2003).
- [119] ESPITI - European User Survey Analysis (ESI-1996-TR95104, November 1996).
- [120] Ralph R. Young, Recommended Requirements Gathering Practices , Northrop Grumman Information Technology, Apr 2002 Issue, the Journal of Defence Software Engineering.
- [121] Karl E. Wiegers, Listening to the Customer's Voice, Software Development, March 1997
- [122] Kavitha Ranganathan and Ian Foster, Identifying Dynamic Replication Strategies for a High- Performance Data Grid, International Workshop on Grid Computing, Denver, 2002.
- [123] B. Tierney, W. Johnston, M. Thompson, A Data Intensive, Distributed Computing Architecture for Grid Applications, Future Generation Computer Systems (an Elsevier Journal), vol 16, #5, April 2000, pp 473-481.
- [124] T. Hagras, J. Janecek, A simple scheduling heuristic for heterogeneous computing environments, Second International Symposium on Publication Parallel and Distributed Computing, 2003. Proceedings, 2003 page(s): 104- 110 ISBN: 0-7695-2069-3
- [125] P. Vicat-Blanc Primet, R. Harakaly, F. Bonnassieux , Grid Network Monitoring in the European Grid Project, International Journal of High Performance Computing Applications, Vol. 18, No. 3, 293-304 (2004)
- [126] Mathis, Semke, Mahdavi & Ott, The macroscopic behaviour of the TCP congestion avoidance algorithm, Computer Communication Review, 27(3), July 1997.
- [127] W. Matthews and L. Cottrell, Achieving High Data Throughput in Research Networks, CHEP 2001, China, 2001.

- [128] Koen Holtman, CMS Case Study of Data Grid  
<http://kholtman.home.cern.ch/kholtman/cmsreqsweb/cmsreqs.html>
- [129] Jewgeni H. Dshalalow, on applications of Little's formula, Journal of Applied Mathematics and Stochastic Analysis, Volume 6, Issue 3, Pages 271-275, 1993.
- [130] Jeff Mausolf, Grid Computing Initiative Grid in action: Managing the resource managers, IBM developerWorks, July 2005 <http://www-128.ibm.com/developerworks/Grid/library/gr-metasched/>
- [131] Joe Meehean and Miron Livny, A Service Migration Case Study: Migrating the Condor Schedd, Midwest Instruction and Computing Symposium, April 2005.
- [132] P. E. Krueger and Miron Livny, A Comparison of Preemptive and Non-Preemptive Load Distributing, Proc. of the 8th International Conference on Distributed Computing Systems, pp. 123-130, June 1988.
- [133] V.Hamscher, U.Schwiegelshohn, A.Streit, R.Yahyapour, Evaluation of Job-Scheduling Strategies for Grid Computing, Workshop Grid 2000 at 7th International Conference on High Performance Computing (HiPC-2000), Bangalore, India, LNCS 1971, pp. 191 - 202
- [134] Gerald Sabin P. Sadayappan, On Enhancing the Reliability of Job Schedulers, High Availability and Performance Computing workshop in conjunction with LACSI 2005, Santa Fe, New Mexico.
- [135] D. Adami, S. Giordana, M. Repeti, M. Coppola, D. Laforenza, and N. Tonellotto Design and Implementation of a Grid Network-Aware Resource Broker, Parallel and Distributed Computing and Networks, 24th IASTED International Multi-Conference 2006, Austria.
- [136] Jang-uk In et al. Policy Based Scheduling for Simple Quality of Service in Grid Computing, 18th International Parallel and Distributed Processing Symposium, 2004.
- [137] Prem Uppuluri, Narendra Jabisetti, Yugi Lee and Uday Joshi, P2P Grid: Service Oriented Framework for Resource Management, IEEE International Conference on Web Services 2005
- [138] Computing Differentiated Services (CDS) model: a proposal for enabling intra-VO fair share and priorities of computing resources in gLite - draft 0.1 (22/03/06)
- [139] CMS, The Computing Project, Technical Design Report, CERN-LHCC-2005-023
- [140] Andrea Caltroni, Andrea FERRARO, G-PBox: A framework for Grid policy management, EGEE Users Forum CERN 2006
- [141] Jeff Mausolf, IBM, Grid Computing Initiative Grid in action: Managing the resource managers, IBM developerWorks, July 2005 <http://www-128.ibm.com/developerworks/Grid/library/gr-metasched/>

- [142] Douglas Thain, Todd Tannenbaum, and Miron Livny, Distributed Computing in Practice: The Condor Experience, Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005
- [143] Paulo Tibério Bulhões, Chansup Byun, Rick Castrapel, Omar Hassaine N1TM Grid Engine 6 Features and Capabilities [http://www.sun.com/cgi-bin/sun/products-n-solutions/edu/cgi-bin/sort\\_list.cgi?type=wp&sort=date](http://www.sun.com/cgi-bin/sun/products-n-solutions/edu/cgi-bin/sort_list.cgi?type=wp&sort=date)
- [144] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA, Ames Research Center, 1996.
- [145] Bart Jacob, How Grid infrastructure affects application design, IBM developerworks July 2003.
- [146] Steve Yalovitsers & Michael Stefano, Grid Computing with a data Grid place, 2002, <http://www.eastmangroup.com/articles/DataGridPland-Whitepaper1.pdf>
- [147] M.P. Papazoglou, Service-oriented computing: concepts, characteristics and directions, Web Information Systems Engineering, 2003. WISE 2003.
- [148] Talia, Trunflo, Toward a synergy between P2P and Grids, IEEE Internet Computing, July-August 2003
- [149] H. Stockinger, F. Donno, G. Eulisse, M. Mazzucato, C. Steenberg. Matchmaking, Datasets and Physics Analysis, Workshop on Web and Grid Services for Scientific Data Analysis (WAGSSDA), IEEE Computer Society Press, Oslo, Norway, June 14, 2005.
- [150] F. Pacini, Job Description Language How To. <http://server11.infn.it/workload-Grid/docs/DataGrid-01-TEN-0142-02.pdf>, Oct. 2003.
- [151] Rene Brun and Fons Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also <http://root.cern.ch/>.
- [152] Brian L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer and Joseph B. Evans, Enabling Network-Aware Applications, 10th IEEE International Symposium on High Performance Distributed Computing (HPDC), August 2001.
- [153] Federico Farchioni, Monte Carlo Simulation of High-Energy Particle Physics, High Performance Computing on Sun Today and Tomorrow, Colloquium, October 5 - 6 2004, Aachen, Germany
- [154] Bharath Rangarajan, The Other Side of Grid Computing -- An EDF Approach, GemStone Systems April 10, 2006 GridToday
- [155] GridNets conference, <http://Gridnets.org/2006/>

- [156] Les Cottrell, TeraPaths: DataGrid Wide Area Network Monitoring Infrastructure (DWMI), <http://www.slac.stanford.edu/grp/scs/net/proposals/iepm-bw/dgnmi-2p.html>
- [157] Mark Leese and Robin Tasker, Grid Network Performance Monitoring, UK e-Science, 2004 All Hands Meeting
- [158] helga case study, Network Performance Measurement and Monitoring, [www.geant2.net/upload/pdf/PUB-06-005\\_JRA1\\_flyer\\_final\\_proof.pdf](http://www.geant2.net/upload/pdf/PUB-06-005_JRA1_flyer_final_proof.pdf)
- [159] The 451 Group: Limitations in Data Management Capabilities Cause Many Large Enterprises to Delay Grid Deployments [http://www.the451group.com/about/press\\_release\\_download.php?tag=gars\\_6\\_pr](http://www.the451group.com/about/press_release_download.php?tag=gars_6_pr)
- [160] C. Steenberg et al., The Clarens Grid-enabled Web Services Framework: Services and Implementation, CHEP 2004 Interlaken Switzerland.
- [161] M. Thomas, et al., JClarens: A Java Framework for Developing and Deploying Web Services for Grid Computing ICWS 2005, Florida USA, 2005.
- [162] Dongyu Qiu and R. Srikant, Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Network, Proceedings of the SIGCOMM 2004, Portland USA.
- [163] I Foster, A. Iamnitchi, On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. Lecture Notes in Computer Science, 2003. Springer Verlag
- [164] R. Les Cottrell et al., Characterization and Evaluation of TCP and UDP-based Transport on Real Networks., Protocols for Fast Long-Distance networks, Lyon, Feb. 2005. Also SLAC-PUB-10996.
- [165] Spiga Daniele, Servo Lileonello, Loris Faina, CRAB Usage and jobs-flow Monitoring, Computing in High Energy Physics (Chep) 2006, Bombay India
- [166] Yaqing Huang Roch Gu'erin, Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger? Proceedings of the 13th IEEE International Conference on Network Protocols 2005 (ICNP 2005), Boston, Massachusetts, USA.
- [167] Jini web page , <http://www.jini.org>
- [168] H.B. Newman, I.C.Legrand, P. Galvez, R. Voicu, C. Cirstoiu, MonALISA : A Distributed Monitoring Service Architecture CHEP03, La Jolla, California, 2003
- [169] S. Lacaprara et al, CRAB: a tool to enable CMS Distributed Analysis, Prepared for Computing in High-Energy Physics (CHEP '06), Mumbai, India, 13 Feb - 17 Feb 2006
- [170] Iperf Bandwidth measurement tool, <http://dast.nlanr.net/Projects/Iperf/>
- [171] H. Newman, J. Bunn, I. Legrand, D. Nae, S. Ravot, X. Su, F. van Lingen, Y. Xia, Ultralight network testbed: Motivations, architectures, and early experiences, In proceedings of the 24th Simposio Brasileiro de Redes de Computadores (SBRC), Curitiba, Brazil, May 29-June 2, 2006

[172] Rob Pennington, Terascale Clusters and the TeraGrid, Proceedings for HPC Asia, Dec 16-19, 2002, pp. 407-413.

[173] Open Science Grid, <http://www.opensciencegrid.org/>

[174] LCG Computing Grid, <http://lcg.web.cern.ch/LCG/>

[175] <http://www-128.ibm.com/developerworks/grid/library/gr-gdg4/>

[176] G. Cawood, T. Seed, R. Abrol, T. Sloan, TGO & JOSH: Grid Scheduling with Grid Engine & Globus. Proceedings of the UK e-Science All Hands Meetings, Nottingham, 2004

[177] William Lee, A. Stephen McGough, John Darlington, Performance Evaluation of the GridSAM Job Submission and Monitoring System, Proceedings of the UK e-Science All Hands Meeting 2005 19th - 22nd September, Nottingham UK

[178] Rashid J. Al-Ali, Kaizar Amin, Gregor von Laszewski, Omer F. Rana, David W. Walker, Mihael Hategan and Nestor Zaluzec, "Analysis and Provision of QoS for Distributed Grid Applications", Journal of Grid Computing, pp 163-182, Volume 2, Number 2 / June, 2004

## Appendix-A Bulk Scheduling Algorithm

The following is the pseudo code for the algorithm to schedule the bulk jobs on the Grid. It works in association with the algorithm discussed in the section 4.6.4. It takes into account all of the points which were discussed earlier to schedule the bulk jobs. It extends the DIANA algorithm to select the best site for a single or homogeneous bunch of jobs.

1. Describe the job(s)
2. If job(s) matches with local site and local site is least loaded
  - 2.1 Check the queue size and number of jobs by  $N = R * W$  //Little's Formula
  - 2.2 If arrival rate is less than Job processing rate
    - 2.2.1 If the local site has the required data
      - 2.2.1.1 submit the job to local site
    - 2.2.2 else
      - 2.2.2.1 go to step 3
- 3 else
  - 3.1 while(all sites(peers) contacted) and required information is retrieved
    - 3.1.1 calculate the network cost (e.g.; bandwidth between execution and submission site)
    - 3.1.2 Calculate the computation cost
    - 3.1.3 Calculate the data transfer cost(data location, data size etc)
    - 3.1.4 Total cost = network cost + computation cost+ Data transfer cost
    - 3.1.5 Calculate the total cost for the candidate execution sites
    - 3.1.6 Calculate the cost for local job execution
    - 3.1.7 If Total Cost less than Local cost
      - 3.1.7.1 submit job to remote site
      - 3.1.7.2 Copy data or read from remote site // which ever is appropriate
    - 3.1.8 else
      - 3.1.8.1 submit Job to local site
      - 3.1.8.2 Copy data or read from remote site //which ever is appropriate
  - 3.2 end while loop
- 4 Once a job submitted // Either local or Remote Site

- 5 Set the Job Threshold //which varies from site to site
- 6 Set the Time Threshold // which varies from site to site
- 7 Push the jobs in the queue
  - 7.1 Do the Data reading or transfer operations //as decided in 3
  - 7.2 Allocate jobs to processing cycles in FCFS order// First come First served
- 8 Arrange Jobs in the queue on the basis of Job and time threshold
- 9 If the number of job(s) from a user in the queue greater than Job Threshold
  - 9.1 allocate the job(s) to one of the lower priority queues
- 10 else
  - 10.1 check the size of the job by its CPU requirements
  - 10.2 If job is shorter than the jobs in the queues
    - 10.2.1 Use SJF algorithm //Shortest Job First
    - 10.2.2 Allocate this job to one of the higher priority queues
  - 10.3 else
    - 10.3.1 Allocate this to comparatively lower higher queues
- 11 while(Job is Submitted)//Either to local or remote site
  - 11.1 If job(s) is in lower priority Queue
    - 11.1.1 after each threshold time
    - 11.1.2 Check the job status
    - 11.1.3 if Job is not in higher priority queue
      - 11.1.3.1 Increase the priority of the job(s) by a fixed number
  - 11.2 If the job(s) from the user crosses the Job Threshold limit
    - 11.2.1 Check the Job Status and if not in the lower priority Queue
    - 11.2.2 reduce the priority of the job(s) from the User by a fixed number
- 12 end while loop
- 13 end function

## **Appendix-B          List of the Publications**

List of the publications relevant to the area of research and published during the course of the research is given below.

1. Ashiq Anjum, Richard McClatchey, Heinz Stockinger, Arshad Ali, Ian Willers, Michael Thomas,, DIANA Scheduling Hierarchies for Optimizing Grid Bulk Job Scheduling, eScience 2006 Amsterdam.
2. Ashiq Anjum, Richard McClatchey, Arshad Ali, Ian Willers, Bulk Scheduling with DIANA Scheduler, IEEE Transactions on Nuclear Science.
3. Ashiq Anjum, Heinz Stockinger, Richard McClatchey, Arshad Ali, Ian Willers, Michael Thomas, Data Intensive and Network Aware (DIANA) Grid Scheduling. Journal of Grid Computing (Accepted).
4. A. Ali, A. Anjum2, J. Bunn, F. Khan, R.McClatchey, H. Newman, C. Steenberg, M. Thomas, Ian Willers, A Multi Interface Grid Discovery System, The 7th IEEE/ACM International Conference on Grid Computing, Grid2006, Barcelona Spain.
5. Ashiq Anjum, Richard McClatchey, Arshad Ali, Ian Willers, Bulk Scheduling with DIANA Scheduler, Computing in High energy Physics (CHEP 2006), Bombay India.
6. Arshad Ali, Richard McClatchey, Ashiq Anjum, Irfan Habib, From Grid Middleware to a Grid Operating System, The 5th International Conference on Grid and Cooperative Computing (GCC 2006) Changsha, Hunan, China 2006.
7. J. Andreeva, A. Anjum et al, Distributed Computing Grid Experiences in CMS, IEEE Transactions on Nuclear Science, 52(4):884-890, Aug. 2005.
8. Arshad Ali, Ashiq Anjum, Tahir Azim, Adeel Zafar, Atif Mehmood ,Harvey Newman, Ian Willers, Richard McClatchey ,Julian Bunn , Waqas-ur Rehman, Resource Management Services for the Grid Analysis Environment, 2005 International Conference on Parallel Processing (ICPP-2005) Oslo, Norway.
9. Michael Thomas, Conrad Steenberg, Frank van Lingen, Harvey Newman, Julian Bunn, Arshad Ali, Richard McClatchey, Ashiq Anjum, Tahir Azim, Waqas ur Rehman, Faisal Khan, Jang Uk JClarens: A Java Framework for Developing and Deploying Web Services for Grid Computing, The 2005 IEEE International Conference on Web Services (ICWS 2005), Florida USA.
10. Frank van Lingen, Julian Bunn, Iosif Legrand, Harvey Newman, Conrad Steenberg, Michael Thomas, Paul Avery, Dimitri Bourilkov, Rick Cavanaugh, Luakik Chitnis, Mandar Kulkarni, Jang Uk In, Ashiq Anjum, Tahir Azim, Grid Enabled Analysis: Architecture, Prototype And Status, Chep2004 Interlaken, Switzerland.

11. Frank van Lingen, Conrad Steenberg, Michael Thomas, Ashiq Anjum, Tahir Azim, Faisal Khan, Harvey Newman, Arshad Ali , Julian Bunn, Iosif Legrand, Collaboration in Grid Environments using Clarens , Collaborate Environments SCI conference (June 20005) ,July 10-13, 2005 - Orlando, Florida, USA
12. Arshad Ali, Ashiq Anjum , Ian Willers, Richard McClatchey, Julian Bunn , Harvey Newman, Atif Mehmood, Predicting Resource Requirements of a Job Submission in Grid Environment, Chep 2004 Interlaken Switzerland.
13. Frank van Lingen, Conrad Steenberg, Michael Thomas, Ashiq Anjum, Tahir Azim, Harvey Newman, Arshad Ali , Julian Bunn, Iosif Legrand , The Clarens Web Service Framework for Distributed Scientific Analysis in Grid Projects (DRAFT) 2005 International Conference on Parallel Processing (ICPP-2005) Oslo, Norway.
14. Arshad Ali, Ashiq Anjum ,Tahir Azim,Yousaf Shah,Harvey Newman,Ian Willers, Richard McClatchey ,Julian Bunn , Saima Iqbal,Tony solomonides, Distributed Heterogeneous Relational Databases Integration Services for An Interactive Scientific Analysis over the Grid, 2005 International Conference on Parallel Processing (ICPP-2005) Oslo, Norway.
15. Conrad Steenberg, Frank van Lingen, Michael Thomas, Harvey Newman, Julian Bunn, Iosif Legrand, ,Richard Cavanaugh, Dmitry Bourilkov, Jang Uk , Arshad Ali,Ashiq Anjum, Tahir Azim, Asif Muhammad, Anzar Afaq, The Clarens Grid-enabled Web Services Framework: Services and Implementation, Chep 2004 Interlaken Switzerland.
16. Arshad Ali, Ashiq Anjum , Ian Willers, Richard McClatchey ,Julian Bunn , Harvey Newman , Atif Mehmood, A Taxonomy and Survey of Grid Resource Planning and Reservation Systems for Grid Enabled Analysis Environment, 2004 International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES),Wuhan, China.
17. Andreeva, A. Anjum et al, Use of Grid Tools to Support CMS Distributed Analysis, In Proceedings of IEEE Nuclear Science Symposium, Rome 2004.
18. Arshad Ali, Ashiq Anjum , Ian Willers, Richard McClatchey ,Julian Bunn , Harvey Newman , Adeel Zafar, Job Interactivity using Steering Service in Grid Enabled Analysis Environment, Chep 2004 Interlaken Switzerland.
19. Arshad Ali,Ashiq Anjum, Michael Thomas, Conrad Steenberg ,Julian J. Bunn, Harvey B.Newman, Tahir Azim,Rizwan Haider, JClarens: A Java Based Interactive Physics Analysis Environment for Data Intensive Applications, 2004 IEEE International Conference on Web Services (ICWS'2004), San Diego California, USA.
20. Arshad Ali, Ashiq Anjum , Ian Willers, Richard McClatchey ,Julian Bunn , Harvey Newman , waqas-ur Rehman, Job Monitoring in Interactive Grid Analysis Environment, Chep 2004 Interlaken Switzerland.

21. Conrad D. Steenberg, Eric Aslakson, Julian J. Bunn, Harvey B. Newman, Michael Thomas, Ashiq Anjum, Web Services and Peer to Peer Networking in a Grid Environment, Proceedings of the 8th International Conference on Advanced Technology and Particle Physics, Milan Italy 2003.
22. Ashiq Anjum, Arshad Ali, Richard McClatchey, Ian Willers, Optimal Replica Location for Efficient Job Scheduling in a Grid Analysis Environment, Prep2005 Lancaster, UK.
23. Arshad Ali, Ashiq Anjum, Fawad Nazir, Tallat Tarar, Hamid Abbas, Process Maturity for Software Project Outsourcing, IADIS International Conference, Madrid, Spain 6-9 October 2004
24. Arshad Ali, Ashiq Anjum, Tahir Azim, Ahsan Akram, Julian J. Bunn, Harvey B. Newman, Michael Thomas, Conrad Steenberg, Performance Assessment of Handheld Devices for Grid Enabled Physics Analysis, 2nd Workshop on Applications of Wireless Communications (WAWC'04), Lappeenranta, Finland.
25. Ahsan Ikram, Arshad Ali, Ashiq Anjum, Conrad Steenberg, Harvey B. Newman, Julian J. Bunn, Michael Thomas and Tahir Azim, Grid Enabled Data Analysis on Handheld Devices, International Networking and Communications Conference 2004(INCC, 2004), LUMS Lahore, Pakistan.
26. Yasir Mehmood, Asif Jan, Arshad Ali, Ashiq Anjum, Umar Kalim, Virtual Data Folder for Seamless Storage and Access of Scientific Data in GRID Environment, 3rd International Symposium on High Capacity Optical Networks and Enabling Technologies, 2006 Charlotte, North Carolina USA
27. Arshad Ali, Ashiq Anjum, Ahsan Akram, Naveed Ahmed, Ian Willers, Richard McClatchey, Julian Bunn, Harvey Newman, Michael Thomas, Conrad Steenberg Distributed Analysis and Load Balancing on Hand-held devices using Multi-Agents Systems for Grid Enabled Analysis Environment, GCC2004, China.
28. Arshad Ali, Ashiq Anjum, Tahir Azim, Ahsan Akram, Julian J. Bunn, Harvey B. Newman, Michael Thomas, Conrad Steenberg, Distributed Analysis on Handheld Devices in Grid Analysis Environment, 13th Ist Mobile and Wireless Communications Summit, 2004, Lyon, France.
29. Ashiq Anjum, Arshad Ali, Tahir Azim, Ahsan Akram, Julian J. Bunn, Harvey B. Newman, Conrad Steenberg, Michael Thomas, Investigating the Role of Handheld devices in the accomplishment of Interactive Grid-Enabled Analysis Environment, Grid and Cooperative computing (CC2003), China.
30. Arshad Ali, Ashiq Anjum, Tahir Azim, Julian Bunn, Ahsan Ikram, Richard McClatchey, Harvey Newman, Conrad Steenberg, Michael Thomas, Ian Willers, Mobile Computing in Physics Analysis - An Indicator for eScience, The Third

International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2006) ,October 2006 , London, U.K.