Dynamic Decentralized Self Organizing Scheduling with DIANA (Data Intensive And Network Aware) Scheduler

By

NC Muhammad Raafay Salam

NC Omer Hussain

PC Umar Ayub

PC Abdur Rehman

Cpt Naeem Iqbal

Submitted to Faculty of Computer Science, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment for the requirements of BE Degree in Computer Software Engineering

March 2008

ABSTRACT

Grid Computing due to its effective sharing of heterogeneous resources, economy and portability has become the center of attention for execution of resource hungry divisible processes. But along with the promising characteristics of grid computing it also poses great challenges in its implementation due to the geographical distant resources owned by individuals with differing access and cost policies.

The vast applications of Grid Computing in the field of e-sciences gave rise to the need for bulk scheduling (i.e. scheduling of a bulk of jobs as a unique entity). Splitting the bulk may result in a very large number of jobs, making it a hideous job for schedulers and also very time consuming in case of centralized schedulers.

In order to exploit the true potential of grid computing workloads need to be scheduled efficiently amongst the participating machines. Centralized schedulers have been implemented to perform the job of load balancing but the grid as a whole lacks the essence of autonomy and self organization. By introducing decentralized schedulers, the limitations such as scalability, posed by centralized schedulers can be tackled. And dependency on a central scheduler is overcome by applying lower level autonomous schedulers. A decentralized approach for bulk scheduling at site level and subgrid level by deploying autonomous site level and low level schedulers is proposed.

DECLARATION

No portion of the work presented in this documentation has been submitted in support of any other award or qualification either at this institution or elsewhere.

DEDICATION

In the name of Allah, the Most Merciful, the Most Beneficent

To the owner of the word ‘Mother’.

From a clot to an engineer she has always been a protector. Thank you for that journey. May the best of heavens befall upon you.

ACKNOWLEDGEMENT

Thanking always the owner of the word ‘Mother’ because without her blessings the following acknowledgements would have never been written.

We are eternally grateful to Almighty Allah for giving us with the strength and determination to undertake and complete the project.

We gratefully recognize the continuous supervision and motivation provided to us by our Project Supervisor, Mr. Bilal (MIS-CELL), without his personal supervision, advice and help, timely completion of this project would have been impossible.

For the completion of this project, we are greatly indebted to Dr. Ashiq Anjum for his continuous guidance, remarkable suggestions, keen interest, friendly discussions and every possible support. He spared a lot of his precious time in advising and helping us

We deeply treasure the unparallel support and tolerance that we received from our friends for their useful suggestions that helped us in completion of this project. We are also deeply obliged to our families for their never ending patience and support for our mental peace and to our parents for the strength that they gave us through their prayers.

A word of thanks to the Military College of Signals (from Commandant to the Staff) as it has been our foundation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Prefaces

Grid computing is expected to provide easier access to remote computational resources that are usually locally limited. Distributed computer systems are joined in such a grid environment in which users can submit jobs that are automatically assigned to suitable resources. The idea is similar to metacomputing where the focus is limited to compute resources. Grid computing takes a broader approach by including networks, data etc. as accessible resources. In addition to the benefit of access to locally unavailable resource types, there is also the expectation that a larger number of resources are available for a single job. This is assumed to result in a reduction of the average job response time. Moreover, the utilization of the grid computers and the job-throughput is likely to improve due to load-balancing effects between the participating systems. Typically the parallel computing resources are not exclusively dedicated to the grid environment. Due to the geographically distributed resources the management of the grid environment becomes rather complex, especially the scheduling of the computational tasks. A typical grid infrastructure is shown in Figure 1.1.

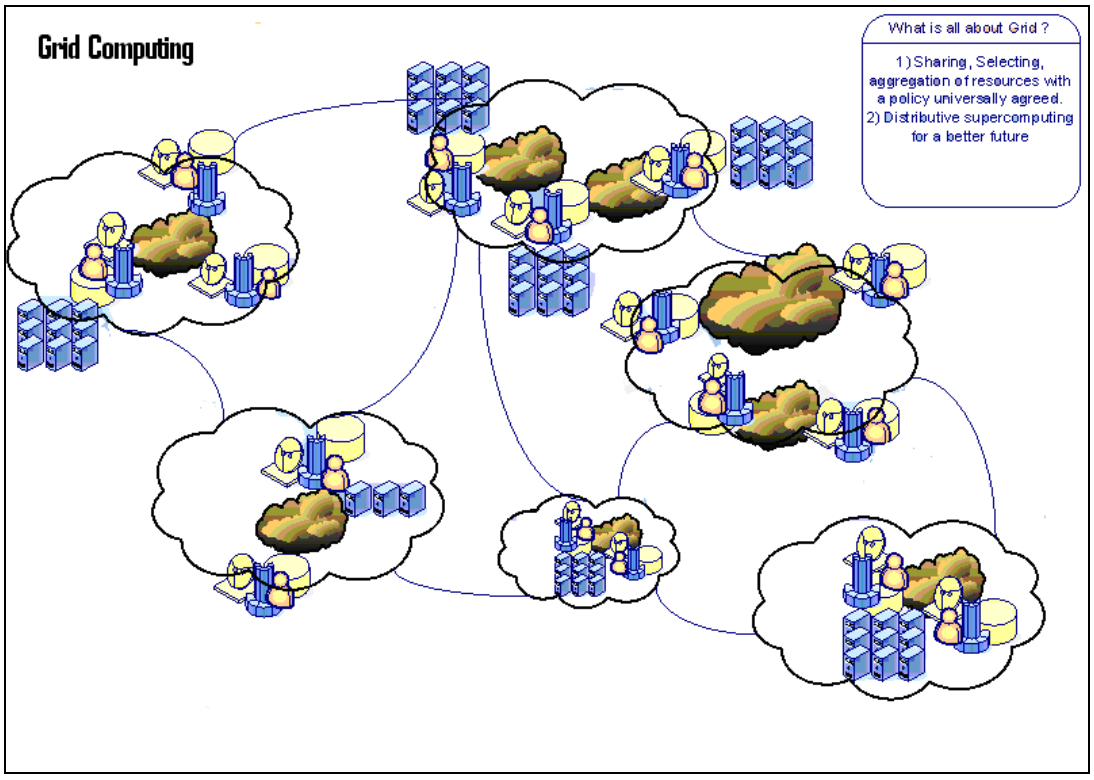

Figure 1.1 A geographically distributed grid

1.2 Scheduling

Resource management is a central task in any Grid system. Resources may include resources such as compute cycles, network bandwidth, and storage systems. Effective resource management and scheduling is a challenging issue, and data location and network load 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 scheduling paradigm since the Grid can be subject to failure if networks do not perform. Similarly, a site with the required data may not be the optimal location to perform the computation if it does not have sufficient available computational resources. All these parameters must be considered in making efficient scheduling decisions.

When a job is submitted to a Grid scheduling system, the scheduling system has the responsibility to select a suitable resource and then to manage the job execution. The decision of which resource should be used is the outcome of a matchmaking process between submission requests and available resources. However, during this matchmaking process, some adaptive scheduling mechanisms are needed, with appropriate heuristics, which can take into account the characteristics of the network to enable efficient scheduling of data intensive jobs to viable computing resources.

1.3 Scheduler types

It has been realized that scheduling is a fundamental issue in achieving high performance on metacomputers and computational grids. In grid environments, there are three types of schedulers to meet different performance goals. Resource schedulers coordinate user requests for accessing a given resource to ensure fairness and to optimize utilization [1]. Application schedulers promote the performance of individual applications by optimizing performance measures such as execution time and speedup [1]. Job schedulers aim to optimize the overall performance of a system, e.g., minimizing the average job response time and maximizing the number of jobs executed in certain period of time. Job scheduling on a metacomputer and grid is very different from job scheduling on a traditional parallel computer due to heterogeneity of communication speed (even though when all the processors are homogeneous). A job is typically divided into sub jobs which are assigned to different machines on a computational grid for execution.

1.4 Problem Description

One important drawback of existing schedulers is that network bottlenecks and execution or queuing delays can be produced in job scheduling. Data intensive applications analyze large amounts of data which are replicated to geographically distributed sites. If data are not replicated to the site where the job is supposed to be executed, the data need to be fetched from remote sites. This data transfer from other sites will degrade the overall performance of the job execution.

Each data intensive application produces different amounts of data. For performance gains in the overall job execution time and to maximize the Grid throughput, it is needed to align and co-schedule the computation and the data (the input as well as the output) in such a way that the overall computation and data transfer cost can be reduced. It may even decide to send both the data and executables to a third location depending on the capabilities and characteristics of the computing, network and storage resources.

1.5 Objective

The objective is to develop meta-scheduler, a process which allows a user to schedule a job across multiple sites. And local schedulers which schedule the jobs on local sites or rootgrid level. As more complexity is added to the Grid, particularly with geographically dispersed sites or nodes.

The following are the intended objectives; to reduce Queue time and waiting time, to lessen Site load and processing time, to minimize Transfer time for data, executables and results.

1.6 The strategy

Data intensive applications often analyze large amounts of data which can be replicated over geographically distributed sites. If the data are not replicated to the site where the job is intended to be executed, the data will need to be fetched from remote sites. This data transfer from other sites can degrade the overall performance of job execution. If a computing job runs remotely, the output data produced needs to be transferred to the user for local analysis. To provide improvements in the overall job execution time and to maximize Grid throughput, the strategy is to align and co-schedule the computation and the data (the input as well as the output) in such a way that the overall computation and data transfer costs can be reduced. It may even decide to send both the data and the executables to a third location depending on the capabilities and characteristics of the available resources.

1.7 Project Descriptions

The system contains the following vital features which are to be known by the user; the system has been implemented on Windows XP platform (due to familiarity of end user). It is recommended that the service be run on minimum Intel based 2.0 GHz microprocessor with minimum of 512 MB RAM since the system is real time system and require massive CPU cycles and network to be reliable .

1.8 Development Environment

The software main module which is in turn divided into many sub modules is made in Python Version 2.5. In addition help and guidance was taken from Python Doc, Twisted Programming and also from the internet.

Adobe Flex programs and Libraries are also used by the software. The methodology followed is Water fall Model. Testing was conducted after the successful completion of each module and in case of an error a bug report was being generated.

1.9 Project Limitations / Constraints

The software was tested and analyze thoroughly and the limits and constraints of the system are: the system is compatible with Windows environment for the time being and can be made to run on Linux by getting resource Ads. For the time being the system resources are being extracted from windows registry and it will be different for Linux.

1.10 Work Breakdown Structure

For the successful completion of the project, the project was divided into main modules and structures. It was ensured that each task being assigned was carried out appropriately and on time.

Table 1.1 Work Breakdown Structure

Requirements Engineering	
Requirements Elicitation	Interaction with Domain Expert . Understanding basis of Grid computing and study related Problems.
Developing Problem Statement	SRS Preparation
Solutions Evaluations	Analyze various options Available (i.e. Platform Compatibility, Language). Propose the best Approach to Problem Solution.
Analysis	Feasibility Study.
Assignment and Planning	Assignment of Tasks to the syndicate members. Preparation of Gantt Charts. Preparation of TimeLine Charts.
Literature Review	Grid computing Research Work. Grid Scheduler Research Work. Condor.

	Remote procedure calls.
Documentation	Preparation of System Manual. Preparation of Data Dictionary. Detailed Thesis. Research Papers and Publications. User Manuals.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

Grid computing is applying the resources of many computers in a network to a single problem at the same time - usually to a scientific or technical problem that requires a great number of computer processing cycles or access to large amounts of data. Grid computing requires the use of software that can divide and farm out pieces of a program to as many as several thousand computers. Grid computing can be thought of as distributed and large-scale cluster computing and as a form of network-distributed parallel processing. It can be confined to the network of computer workstations within a corporation or it can be a public collaboration (in which case it is also sometimes known as a form of peer-to-peer computing).

In the past two decades, numerous scheduling and load balancing techniques have been proposed for locally distributed multiprocessor systems. However, they all suffer from significant deficiencies when extended to a Grid environment: some use a centralized approach that renders the algorithm unscalable, while others assume the overhead involved in searching for appropriate resources to be negligible.

2.2 Metacomputing

Metacomputing is all computing and computing-oriented activity which involves computing knowledge (science and technology) common for the research, development and application of different types of computing. Metacomputing includes: organization of large computer networks, choice of

the design criteria (e.g. peer-to-peer or centralized solution) and metacomputing software (middleware, metaprogramming) development where, in the specific domains [2]. The concept metacomputing is used as a description of software meta-layers which are networked platforms for the development of user-oriented calculations [3], for example for computational physics and bio-informatics.

Metacomputing allows scientists and engineers to coordinate computational grids capable of sharing and analyzing complex simulations and data.

2.3 Grids versus conventional supercomputers

The ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across 'multiple' administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements [4].

"Distributed" or "grid" computing in general is a special type of parallel computing which relies on complete computers (with onboard CPU, storage, power supply, network interface, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus [4].

The primary advantage of distributed computing is that each node can be purchased as commodity hardware, which when combined can produce similar computing resources to a multiprocessor supercomputer, but at lower cost. This is due to the economies of scale of producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers. The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors.

The high-end scalability of geographically dispersed grids is generally favorable, due to the low need for connectivity between nodes relative to the capacity of the public Internet. Conventional supercomputers also create physical challenges in supplying sufficient electricity and cooling capacity in a single location. Both supercomputers and grids can be used to run multiple parallel computations at the same time, which might be different simulations for the same project, or computations for completely different applications. The infrastructure and programming considerations needed to do this on each type of platform are different, however.

There are also some differences in programming and deployment. It can be costly and difficult to write programs so that they can be run in the environment of a supercomputer, which may have a custom operating system, or require the program to address concurrency issues.

2.4 Remote Procedure Calls (RPC)

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports [5].

An RPC is analogous to a function call. Like a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure. Figure 2.1 shows the flow of activity that takes place during an RPC call between two networked systems. The client makes a procedure call that sends a request to the server and waits. The thread is blocked from processing until either a reply is received, or it times out. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. RPC specifically supports network applications.

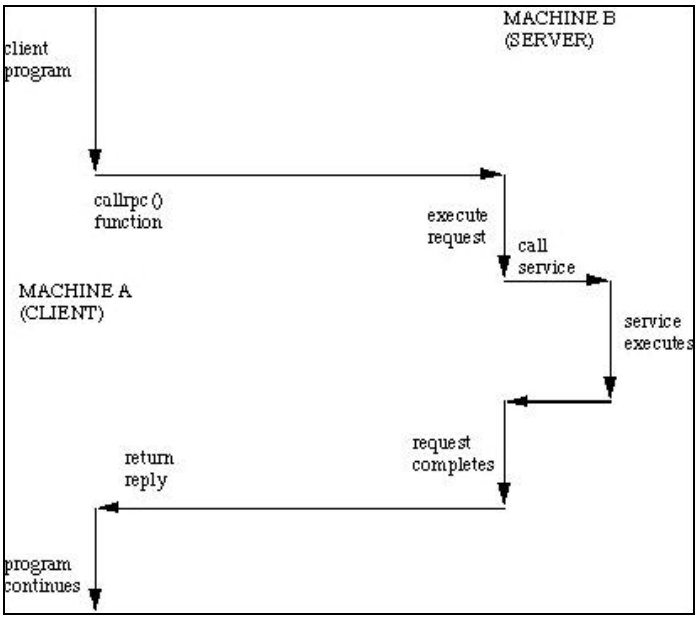

Figure 2.1 Flow of activity that takes place during an RPC

2.5 Local Scheduler Layer Condor

Condor is a specialized job and resource management system (RMS) for compute intensive jobs. Like other full-featured systems, Condor provides a job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their jobs to Condor, and Condor subsequently chooses when and where to run them based upon a policy, monitors their progress, and ultimately informs the user upon completion [6].

In the Figure 2.2 An agent (A) is shown executing a job on a resource (R) with the help of a matchmaker (M).

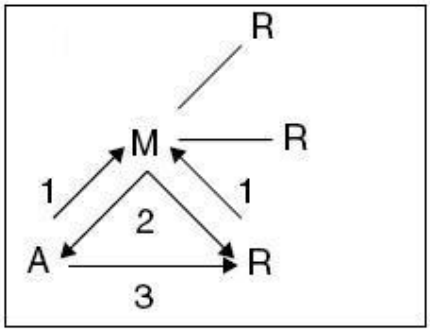

Figure 2.2 Condor pool

The agent and the resource advertise themselves to the matchmaker. Step 2: The matchmaker informs the two parties that they are potentially compatible. Step 3: The agent contacts the resource and executes a job. Each of the three parties – agents, resources, and matchmakers – are independent and individually responsible for enforcing their owner's policies. The agent enforces the submitting user's policies on what resources are trusted and suitable for running jobs. The resource enforces the machine owner's policies on what users are to be trusted and serviced. The matchmaker is responsible for enforcing community policies such as admission control. It may choose to admit or reject participants entirely on the basis of their names or addresses and may also set global limits such as the fraction of the pool allocable to any one agent. Each participant is autonomous, but the community as a single entity is defined by the common selection of a matchmaker.

2.6 Programming Language Selection

Python Programming Language is chosen as implementation language for this project. The prime reasons of this choice over C++, and Java are:

2.6.1 Interpreted Language

In an interpreted environment, the instructions are executed immediately after parsing. Advantages of interpreted languages include relative ease of programming (since once you type your instructions into a text file, the interpreter can run it) and no linker is required. Interpreted languages give you a much quicker development cycle, especially on big programs. There is no doubting that it's simply a tradeoff of execution speed vs. productivity.

2.6.2 Better code readability

Since the project is a part of the Grid Community, people interested to make further progress in the field of Distributed Schedulers might use it. The Object Oriented Approach provided by Python is used. Since it's a rule of thumb to indent code in Python so the code doesn't need any formatting, and the hassle of finding braces for blocks of code has been eliminated by Python.

2.6.3 Extensive and easy to use Socket libraries

Creating a Server Socket is a four step processes in Python programming language, which is quite small as compared to ones in C++, and Java. It is illustrated in Figure 2.3.

```
Step 1:
socket = socket.socket( family, type )
Step 2:
socket.bind( (HOST,PORT) )
Step 3:
socket.listen( backlog )
Step 4:
connection, address = socket.accept()
```

Figure 2.3 Python Server Socket creation

Creating a Client Socket is even simpler and requires just two steps illustrated in Figure 2.4

```
Step 1:
socket = socket.socket( family, type )
Step 2:
Socket.connect((HOST,PORT))
```

Figure 2.4 Python Client Socket creation

2.6.4 Easy to use File Handling Libraries

Since the Project Involves too much file handling and file transfers. This is amongst the prime reason for us to choose Python. Since its pointer free unlike C++, and requires no libraries. An example in Figure 2.5 should demonstrate this ease of use.

```
Python Code:
      #no library imports!
      file = open( "C:\\test.dat", "w" ) # open file in write mode
      file.write( “Hello World!”) # write string ‘Hello World’ to file
      file.close( ) # close the file

C++ Code:
      #include <iostream.h>
      Ofstream file
      file.Open ("c:\test.dat");
      file << "Hello World!" << endl;
      file.close()
```

Figure 2.5 File Handling Python vs. C++

2.6.5 Simple but dynamic data structures like Lists, and Tuples

These data types enabled us to accomplish complex tasks through minimal lines of code. Strings, lists and tuples are all sequences—a data type that can be manipulated through indexing and “slicing.” Although lists are not restricted to homogeneous data types (i.e., values of the same data type), lists to store sequences of homogeneous values i.e. Machine resource ads, Job Objects and etc are used.

2.6.6 Simple access to Win32 API’s

For the time being the version of the Scheduler is Windows Dependent since it uses Win32 API’s to fetch CPU Utilization, Memory Usage, and Network Performance. These API’s were very easily available at the Python’s Official Website for download.

2.6.7 Compact and Platform Independent

Python is one of the most highly portable programming languages in existence. Originally, it was implemented on UNIX, but has since spread to many other platforms, including Microsoft windows and Apple Mac OS X. Python programs often can be ported from one operating system to another without any change and still execute properly.

2.6.8 Three step Remote Object Registration / One Step Remote Object Access

XML/RPC a Remote Procedure Calling facility provided within python is used. Other RPC's facilities include SOAP and WSDL. But the quickest of all was XML/RPC because it dint embed any extra information about the remote object during end-end communication whereas SOAP and WSDL do. This extra information is regarding the description of the object which isn't entirely necessary in this case. Compared to facilities like RMI and CORBA in Java. XML/RPC is much simple and easy to use. For remote object registration only three steps are required which is illustrated in Figure 2.6.

```
RPCServer = SimpleXMLRPCServer((HOST, 8500))

RPCServer.register_instance(RPCs(self.mfq,self.directory))

RPCServer.serve_forever()
```

Figure 2.6 Remote Procedure Calling in Python

2.6.9 Multicasting Libraries

Multicast techniques are also required in the project for Discovery Services @ Meta Scheduler Level and for Resource Advertisements @ Local Scheduler level.

A library provided by twisted community for download at www.twistedmatrix.com is used for multicasting mechanisms in the project. It's a very simple step by step mechanism that is far simpler than one provided by Java.

2.7 Summary

As the project is the research project, which requires massive study of topic and all the previous work done in this area, many research papers and guides were studied which helped us understand the domain history of past work. As a research project, studies continue throughout the project, but major phase was completed in the early months. Tools and software like Condor, Globus, Suse, Adobe Flex and Python are also explored for the purpose.

Chapter 3

SCHEDULING OPTIMIZATION ALGORITHM

3.1 Introduction

In this chapter the main phase for scheduling on a Grid is discussed. That is Resource Discovery (generating a list of potential resources),. The following input parameters are required for a network-aware scheduling and matchmaking algorithm data transfer cost, hard disk space, computing cycles available and Site loads.

3.2 Cost Estimators

There are three major cost estimates which need to be calculated for the scheduling Algorithm: network, computation and data transfer cost.

3.2.1 Network Cost

First and foremost is the network cost which depends on many individual parameters. The load, capacity and availability of network links used during data transfers may heavily affect the Grid application performance [7]. Application usage of the network often requires near-real-time, or even real-time, information feedback on the available resources and intelligent decisions on how best to take advantage of these resources [8]. In order to provide the right quality of service 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. TCP throughput can be obtained by combining

the losses and the Round Trip Times (RTTs) using Mathis's formula [9] for deriving the maximum TCP throughput. Given the historical measurements of the packet loss and RTT, the maximum TCP bandwidth for a certain amount of time for various groups of sites can be calculated. Paper [9] describes a short and useful formula for the transfer rate:

$$Rate < \left(\frac{MSS}{RTT}\right) \times \left(\frac{1}{\sqrt{loss}}\right) \quad \text{..... 3.1}$$

Where Rate is the TCP transfer rate, MSS is the maximum segment size, RTT is the round trip time (as measured by TCP), loss is the packet loss rate.

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 quality of service is to measure the number of packets being dropped .i.e. "packet loss". 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). The RTT says nothing about how much information a server site can send in a given period. Moreover, for better quality of service and network predictability, including the jitter in the scheduling algorithm is also necessary. Overall, as the network utilization increases, the number of dropped packets and the amount of jitter also increases. Consequently, the network cost is the combination of all of the above parameters. Weights are assigned to each value depending on the

importance of the parameters in calculating an aggregate value of the network cost (NetCost).

$$NetCost \propto \frac{Losses}{Bandwidth} \quad \text{.....3.2}$$

Where:

$$Losses = RTT \times W1 + Loss \times W2 + Jitter \times W3$$

Where Wi is the weight assigned to each parameter depending on the importance of the particular parameter. Weights are uniformly assigned subject to a cost; a higher cost indicates the importance of that parameter in the scheduling decision and in some cases these weights can be manipulated to prioritize particular parameters in the algorithm [10]. For example, increasing the weight associated with network cost would bias in favor of data intensive scheduling. To tend towards compute intensive jobs, the compute cost weights can be increased. 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. Significance of this parameter can be increased, if required, by assigning a higher value to its associated weight. Higher bandwidth reduces the cost of data transfer and hence the job execution. This behavior can be accommodated by assigning a higher value to its weight W_i. Moreover, jitter is of less importance for data intensive applications and has no significant cost involved due to a higher or lower jitter. A minimal weight can be assigned to this parameter but cannot ignore it completely since if this parameter has a value more than an acceptable figure, then there is some bottleneck involved and this element should then have a higher value to reflect this in the overall scheduling algorithm. The same is the true for the packet loss: a higher packet loss implies a less reliable network,

and less importance should be given to such a connected site when making scheduling decisions.

There are a number of issues which can influence the scheduling decisions from the network point of view and can lead to skewed results. Selecting the best source from which to copy the data requires a prediction of future end-to-end path characteristics between the destination and each potential source. An accurate prediction of the performance obtainable from each source requires the measurement of available bandwidth (both end-to-end and hop-by-hop), latency, loss and other characteristics which are important in file transfer performance. Because network characteristics are highly dynamic, each reported observation must be attributed with timing information, indicating when the observation was made. Route flaps or other instabilities mean that the same end-to-end traffic may experience a completely different environment from moment to moment.

3.3.2 Computation Cost

The second important cost which needs to be part of the scheduling algorithm is that of computation cost. Paper [11] describes a mathematical formula to compute the processing time of a job. It is based on Little's theory.

$$Computaion\ Cost = \frac{Qi}{Pi} \times W5 + \frac{Q}{Pi} \times W6 + SiteLoad \times W7 \quad \text{.....3.3}$$

Where Q is the total number of the waiting jobs on all the sites, Qi is the length of the waiting queue on the site i, Pi is the computing capability of the site i and SiteLoad is the current load on that site. SiteLoad is calculated by dividing the number of jobs in the queue by the processing power of that site. The Qi/Pi ratio computes the processing time of the job. The Qi/Pi 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. Again W5, W6 and W7 are the weights which can be assigned depending on the importance of the queue and the processing capability. For example, a larger queue makes a site less attractive for job placement so it is assigned a higher weight to make the cost higher. Similarly, site load reflects the current load on a site, so again a higher weight is assigned if the load on that site is higher.

It is a challenging task to calculate and predict the dynamic nature of the resources and changing loads on the Grid. The load prediction at a site must be dynamic in nature and the least loaded site at one moment can become overloaded the next moment due to bulk submission. Since a non pre-emptive mode of execution is used, once a jobs gets a CPU job cannot be aborted and moved to other site.

3.3.3 Data Transfer Cost

The third most important cost aspect in data intensive scheduling is the data transfer cost which includes input data, output data and executables. Reference [12] describes a mathematical technique to calculate the aggregate data transfer time which includes all three parameters. Here, bandwidth only is not used to calculate the data transfer cost, rather the network cost is used as calculated in Section 3.3.1. The case of remote data and different remote execution sites is taken so that the metascheduler can consider a worse-case scenario in scheduling.

Data Transfer Cost (DTC) = Input Data Transfer Cost + Output Data Transfer Cost

+ Executables Transfer Cost……..3.4

$$W_8 \times ID \times NC_{(i,j)} + W_9 \times (AD + OD) \times NC_{(local \to j)} + W_{10} \times (N_{(j)} \times (ID + AD) + OD) \times NC_{(j)}$$

Where ID is Input Data, AD is Application Data, OD is Output Data and NC = Network Cost and i and j indicate a certain site

Here, the three different costs for data transfer are discussed. Input data transfer cost is the most significant due to expected large data transfers. Higher network cost will increase the data transfer cost and vice-versa, and the associated weight is used to adjust the value according to its importance. The same is the case for the output data since output data needs to be transferred to the location from where the job was submitted. Application data are executables and user code which will be submitted for execution but might be low compared to the input and output data transfer costs.

The response time can be reduced by moving input data from one site to another 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. 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 in site j and the resulting data will be transferred to the local site.

3.3.4 Total Cost

The total cost is simply a combination of these individual costs as calculated in Sections 3.3.1, 3.3.2 and 3.3.3:

Total Cost C = Network Cost + Computation Cost + Data Transfer Cost…….3.5

The main optimization problem that needs to be solved 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. To simplify the optimization problem it is assumed that any given site can have one or many storage resources (Storage Elements, SEs) or one or many computing resources (Computing Elements, CEs). Therefore, main interest is in the wide-area network performance rather than specifying all network details within a site. It is assumed that the local network latency is roughly homogeneous for all nodes (storage or computing) within a site [10]. The cost of the job placement can be calculated 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. Next, a cost matrix can be populated with cost values against each site. In detail, the number of possible sites are investigated and the total cost calculated for each pair (site i – site j) and put that into cost matrix.

Table 3.1 shows an example cost matrix giving the overall cost of job submission from one site to all four others in the Grid. Cij is the total cost of a particular site I from any other one j in the Grid.

Table 3.1 The cost matrix for five example sites

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

3.4 Conclusion

These costs are the core elements of the DIANA Scheduler in selecting the optimal site for job execution. The network cost calculated in the algorithm is used to select the best replica of a dataset which will be used as input to the scheduler.

Chapter 4

DIANA SCHEDULING

4.1 Introduction

In this section the scheduling strategy is discussed of moving data to jobs or both to a third location. Not only need the network characteristics are used while aligning data and computations, it is also need to optimize the task queues of the meta-scheduler on the basis of this correlation. As a consequence network characteristics can play an important role in the matchmaking process and on Grid scheduling optimization. Therefore a complex scheduling algorithm is required that should consider the job execution, the data transfer and their relation with various network parameters on multiple sites.

4.2 Data intensive and network aware

Data intensive applications often analyze large amounts of data which can be replicated over geographically distributed sites. If the data are not replicated to the site where the job is intended to be executed, the data will need to be fetched from remote sites. This data transfer from other sites can degrade the overall performance of job execution. If a computing job runs remotely, the output data produced needs to be transferred to the user for local analysis. To provide improvements in the overall job execution time and to maximize Grid throughput, it is need to align and co-schedule the computation and the data (the input as well as the output) in such a way that the overall computation and data transfer costs can be reduced. It may even be decided to send both the data and the executables to a third location

depending on the capabilities and characteristics of the available resources. It is not only need to use the network characteristics while aligning data and computations, but it is also needed to optimize the task queues of the Meta-Scheduler on the basis of this correlation since network characteristics can play an important role in the matchmaking process and on Grid scheduling optimization. Thus, a more complex scheduling algorithm is required that should consider the job execution, data transfer and their correlation with various network parameters on multiple sites. There are three core elements of the scheduling problem which can influence scheduling decisions and which needs to be tackled: data location, network capacity/quality and available computation cycles.

4.3 P2P Meta scheduler

In DIANA, independent Meta-Schedulers are not used but instead use a set of Meta-Schedulers that work in a peer-to-peer (P2P) manner. As shown in Figure 4.1, each site has a Meta-Scheduler that can communicate with all other Meta-Schedulers on other sites.

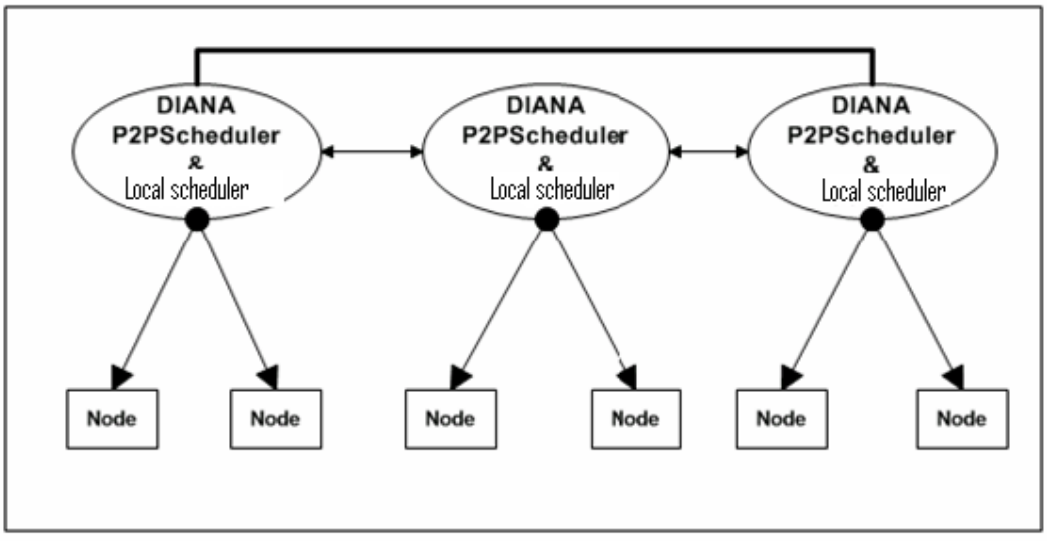

Figure 4.1 Communication between instances of Meta Schedulers

The Meta-Scheduler is able to discover other Schedulers with the help of a discovery mechanism [13]. DIANA Scheduler is a layer over each local scheduler so that these local schedulers can talk directly to each other instead of getting directions from a secondary central manager. In the DIANA architecture each local Scheduler has a local queue plus a global queue which is managed by the DIANA layer. This leads to a self organizing behavior which was missing in the client server architecture.

4.4 RootGrid to RootGrid and RootGrid to Nodes Communication

The nodes are divided into SubGrids, each SubGrid having its own RootGrid. Roughly each site has one RootGrid and may have one or more SubGrids. The Meta-Scheduler works at the RootGrid (Master node) level. The RootGrid to RootGrid communication is in essence a P2P communication between the Meta-schedulers. Each RootGrid maintains a table of entries about the status of the nodes which is updated in real time when a node joins or leaves the system. 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 machine within a SubGrid communicates only with the Meta-scheduler, which itself communicates with the Meta-schedulers at other RootGrids. Consequently, this approach is not just all-to-all communication. This approach shown in the Figure 4.2

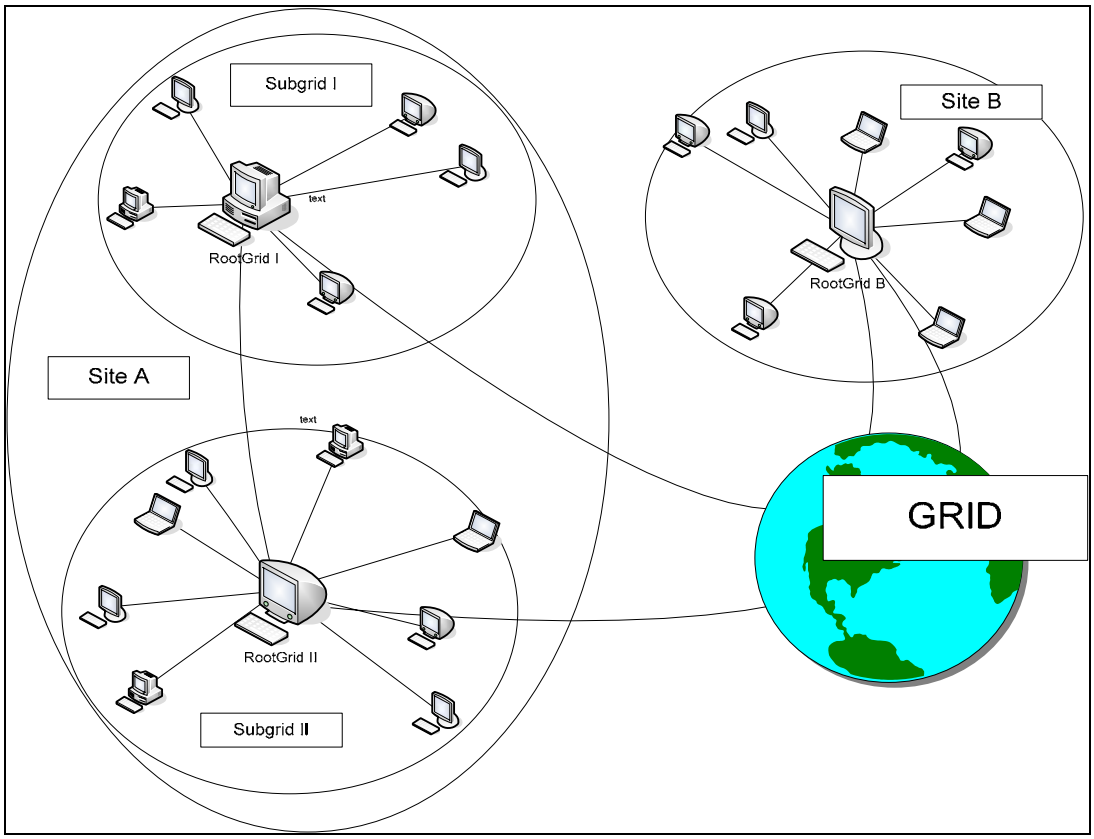

Figure 4.2 Meta scheduler communication mechanism

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 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 standard criteria. Whenever a site becomes part of the Grid, a separate SubGrid encompassing the site resources is created which joins the

nearest RootGrid. If the site is fairly small in terms of the 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.

4.5 General Architecture

The overall architecture of the DIANA Scheduler is shown in Figure 4.3. It includes a DIANA meta-scheduler with its internal matchmaker. The meta-scheduler uses network information to make optimal scheduling decisions.

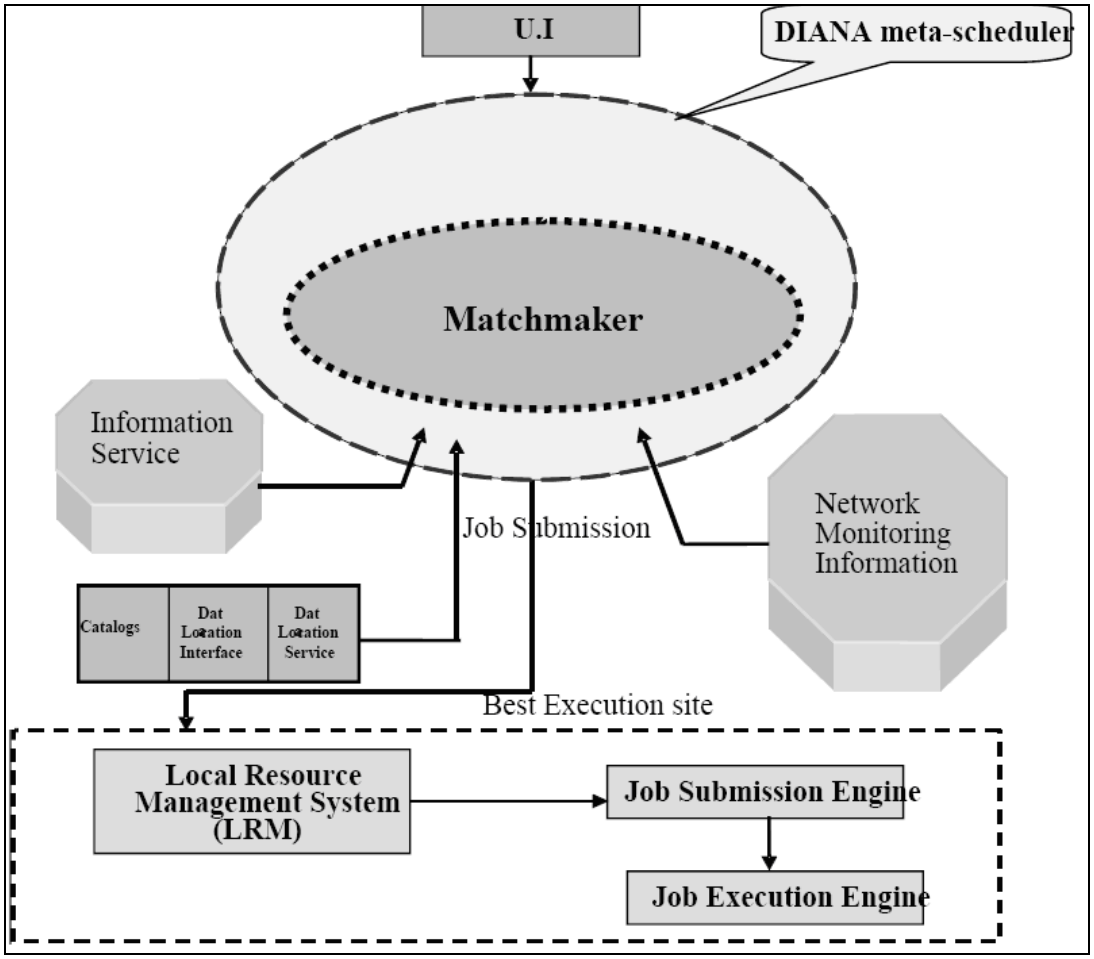

Figure 4.3 DIANA Scheduler and the Discovery Service

The Data Location Service makes use of the Data Location Interface [14] to find the list of the dataset replicas and then uses network statistics to find the “best” replica which is then used by the scheduler. The DIANA scheduling system is implemented as a peer to peer system as discussed in [15]. It should also be noted that an external job submission/execution system needs to be used since the DIANA meta-scheduler only provides scheduling information but does not take care of the actual dispatching/submission of the job to a local resource management system.

The DIANA meta-scheduler mediates between data providers and data requesters. The first step, which is to discover the available resources, is defined as resource discovery in Chapter 3. A resource 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, if the value of this attribute satisfies the condition “Value(processing power) > 2 GHz”. If the request can be successfully satisfied, the matchmaker responds with a list of ranked resources. After this, the scheduling optimization algorithm is used to select the best resource and a job is subsequently scheduled to be executed on this resource. The DIANA meta-scheduler keeps track of the load on the sites and selects a site which has a minimum load and queue and has the desired data, processing capability and network stability. 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. The database collects the historical as well as real time information to obtain a current and previous view of the system state.

4.6 The Scheduling algorithm

This Scheduler deals with both computational jobs as well as data intensive jobs. In the DIANA Scheduling scheme, the Scheduler consults its peers, collects information about the peers including network, computation and data transfer costs and selects the site having minimum cost. To schedule computational jobs, this algorithm selects resources which provide most computational capability. The same is the case with data intensive jobs. To schedule data intensive jobs, it is needed to determine those resources where data can be transferred cost effectively. Since the different costs are calculated, these costs can be brought together under a scheduling algorithm as described below.

In the case of a computational job, more computational resources are required and the algorithm should schedule a job on the site where the computational cost is a minimum. At the same time, the job's files need to be transferred so it is needed to be ensured that the job can be transferred as quickly as possible. Therefore, the Scheduler will select the site with minimum computational cost and minimum transfer cost. In the case of a data intensive job, the preferences will change. In this case the job has more data and less computation and it is needed to determine the site where data can be transferred more quickly and at the same time, where computational cost is also a minimum. The algorithm keeps on scheduling until all jobs are submitted. After every job the cost to submit the next job is calculated.

4.6.1 Algorithm for a Compute Intensive Job

If the job is compute intensive then the algorithm in Figure 4.4 depicts the scheduling behavior of DIANA.

```
computationCost[] = getAllSitesComputationCost();
NetworkCost []= getAllSitesNetworkCost();
arrageSites[] = SortSites(computationCost, NetworkCost); //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
```

Figure 4.4 Algorithm for Compute Intensive Job

4.6.2 Algorithm for a Data Intensive Job

If the job is data intensive then the algorithm in Figure 4.4 depicts the scheduling behavior of DIANA.

```
dataTransferCost[] = getAllSitesDataTransferCost();
NetworkCost []= getAllSitesNetworkCost();
arrageSites[] = SortSites ( dataTransferCost, NetworkCost ); //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
```

Figure 4.5 Algorithm for Data Intensive Job

4.6.3 Algorithm for a Compute and Data Intensive Job

Algorithm in Figure 4.6 depicts the scheduling behavior of DIANA.

```
computationCost[] = getAllSitesComputationCost()
dataTransferCost[] = getAllSitesDataTransferCost()
NetworkCost []= getAllSitesNetworkCost();
// since length of computationCost and dataTransferCost array is
same. So any of them can be used
siteTotalCost [] = new Array[computationCost.length]
for i = 1 to computationCost.length
siteTotalCost [i] = computationCost[i] + dataTransferCost[i] +
NetworkCost [i]
end loop
sites [] = SortSites(siteTotalCost)
for j = 1 to sites.length
site = sites[i]
if ( site is alive)
schedule the job to this site
end loop
```

Figure 4.6 Algorithm for a Data and Compute Intensive Job

4.7 Priority based Scheduling

The scheduling algorithm is a priority based 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. Priorities can be defined either internally or externally. Internally defined priorities use some measurable 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 [16]. 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 preemptive. The

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. Since most jobs are data intensive, this makes it increasingly important to consider the non pre-emptive mode as a primary approach.

4.8 Job Migration Algorithm

To illustrate job migration let us take an example scenario where a user submits a job to the Scheduler and the Scheduler puts this job into queue management. If the queue management algorithm of the Scheduler decides that this job should remain in the queue, it may have to wait a considerable time before it gets serviced or before it is migrated to some other site. In this case the queue management module will ask the scheduling module to migrate the job.

4.8.1 Peer Selection Criteria

The important point to note here is that the job must be scheduled at that site where it can be serviced earliest. Therefore the peer selection criteria are based on two things, which are minimum queue length and the minimum cost to place this job on the remote site. The Scheduler will communicate with its peers and ask about their current queue length and the number of jobs with priorities greater than the current job's priority. The site with 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 depicted by Figure 4.7.

```
Sites[] = GetPeerList( )
int count = Sites.length // total no of sites
int queueLength [ ] = Sites.length
int jobsAhead[]= new int[ count ]
for ( i=1 to count )
jobsAhead [i] = getJobsAhead( site[i] )
end for
find the peer with minimum jobsAhead
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
```

Figure 4.7 Peer Selection Algorithm

4.8.2 Communication for job migration

First of all scheduler will get the information about the available peers from the discovery or information service. Then it will communicate with each peer and collect the peer's queue length, total cost, and the number of jobs 'ahead' of the current job's priority. After this, it will find out the site with the minimum queue length and minimum jobs ahead. If the number of jobs and total cost of the remote site is more than the local cost, then this job is scheduled to the local site. In this case the other sites are already congested and there is no need to migrate the job. Therefore that job will remain in the local queue and will be served when it gets the execution slot on the local site. Otherwise the job is moved to a remote site subject to the cost mechanism. This decision is made on the principle that this job as a result will get quicker

execution since the targeted site has overall least cost and least queue as compared to other sites.

4.9 Summary

Considering all the costs data transfer cost, network cost and availability of resources the design of meta-scheduler is such that which perform the match making job and select the best peer for the submitted jobs. The meta-scheduler communication mechanism is also discussed which concludes that rootgrid communicate with its local nodes and meta-scheduler communicates with the meta-scheduler on the other sites. So in this mechanism not all the nodes are communicating with each other and it is not just all to all communication

Chapter 5

CLUSTER LEVEL SCHEDULER

5.1 Introduction

This section is aimed to discuss the need for a 'Decentralized Cluster Level Scheduler and its implementation. A cluster can be thought of as a group of homogeneous computers that are connected through a LAN. Two major architectures that exist for Schedulers are:

5.1.1 Centralized architecture

The historic architecture for Job scheduling software. The Job Scheduling software is installed on a single machine (Master) while on production machines only a very small component (Agent) is installed that awaits commands from the Master, executes them, and returns the exit code back to the Master.

5.1.2 Cooperative architecture

A decentralized model where each machine is capable of helping with scheduling and can offload local jobs to other cooperating machines. This enables dynamic workload balancing to maximize hardware resource utilization and high availability to ensure service delivery.

5.2 Effect of Decentralized Schedulers

Till now the execution of DIANA (Metascheduler) is discussed over a centralized cluster level scheduler i.e. Condor in this case. But imagine the speed at which the grid would process jobs if it is completely decentralized by

replacing the centralized cluster level scheduler with a decentralized one. This was the next milestone after completing the designed and development of the meta-scheduler. This milestone in the project has also been successfully completed.

Though it is not as complex as Condor, but for simple batch Jobs it is far more faster than Condor. The few basic parameters to be parsed out of the Job Description File are just included; whereas the Condor Application supports many parameters. The Cluster Level Scheduler can so far execute simple batch files and java class files and can handle input files as well. In this chapter following aspects of the Decentralized Cluster Level Scheduler will be highlighted; Architectural Design, Class Diagrams, and Sequence Diagrams

5.3 Architectural Design

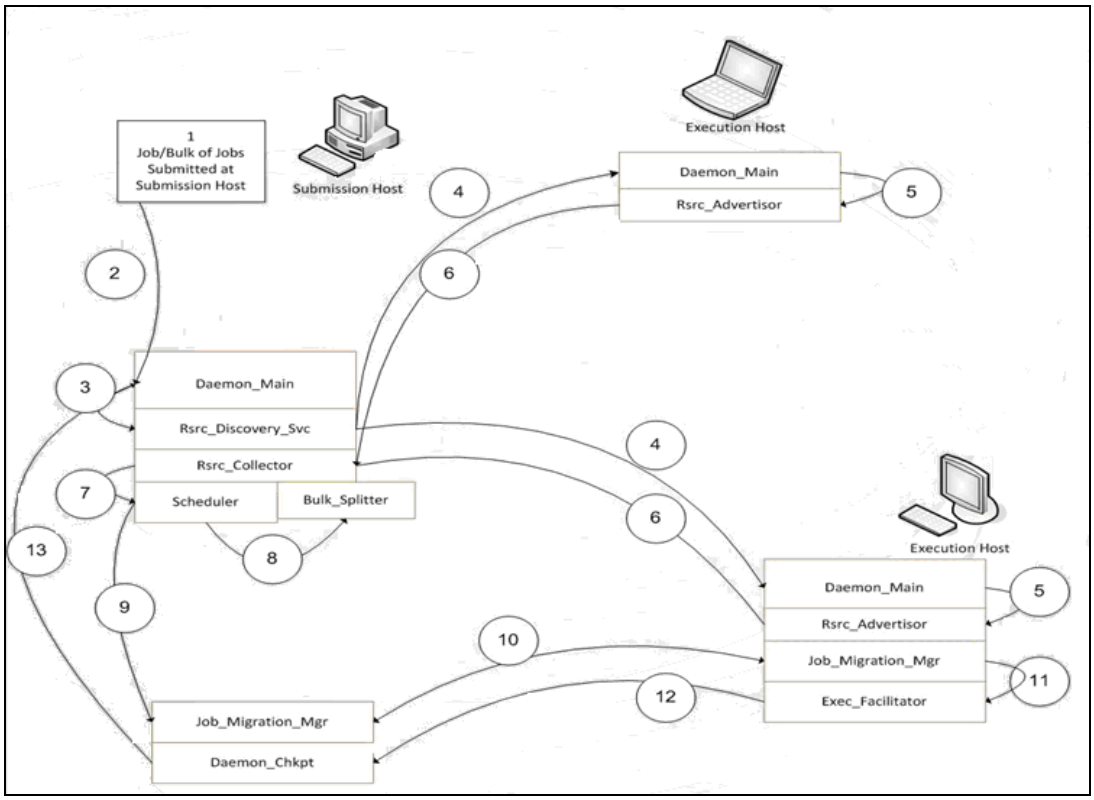

Figure 5.1 Architecture design of the Local scheduler

Job/Bulk of Jobs Submitted at Submission Host. Job transferred to the Daemon_Main to be scheduled remotely if its resource requirements exceed local resources. Daemon_Main calls Resource_Discovery_Svc. Rsrc_Discovery_Svc broadcasts requests for resource advertisements. Daemon_Main calls Resource_Advertisor. Rsrc_Advertisor of each node returns Resource Advertisements to Rsrc_Collector of Submission Host. Rsrc_Collector passes Resource Advertisements to Scheduler to map jobs to resources. Scheduler Calls Bulk_splitter on bulk job if it is to large to be mapped onto any available resources. Mapped Jobs sent to Job_Migration_Mgr. Job_Migration_Mgr dispatches jobs to appropriate Hosts. Received Job sent to Exec_Facililtator Exec_Facililtator executes job and returns temporary output periodically to Daemon_Chkpt. Daemon_Chkpt saves checkpoint information till final output has been received and calls Daemon_Main. Daemon_Main finalizes and performs house cleaning actions.

5.4 Class Diagram and Description of Classes

Figure 5.2 shows a class diagram and their description of Cluster Level Scheduler. This easily illustrates the abstract implementation of the scheduler.

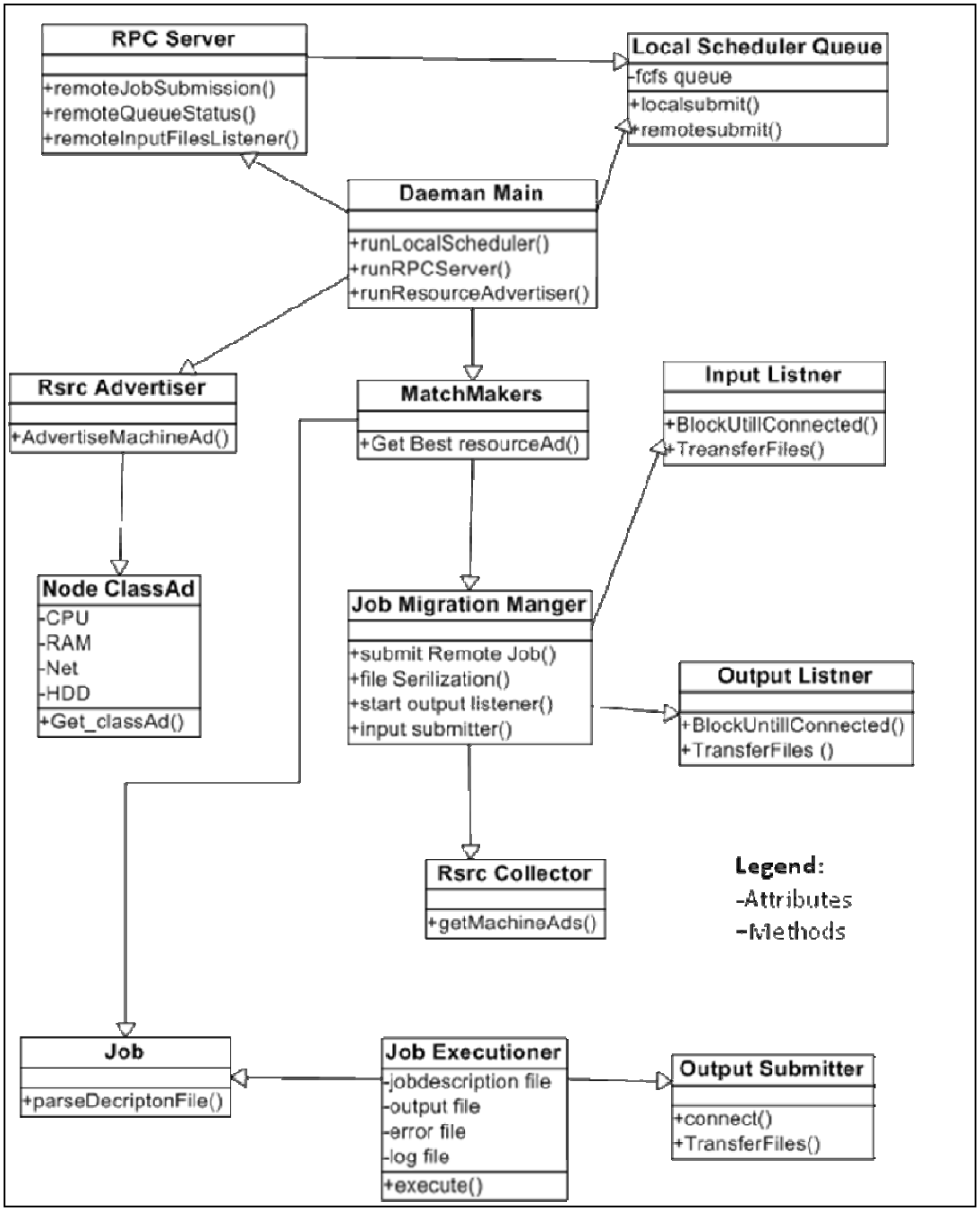

Figure 5.2 Class Diagram and Description of Classes of local scheduler

5.5 Local Execution Sequence Diagram

Figure 5.3 depicts the sequence of actions that the local scheduler would take when a job is scheduled locally for execution. Figure 5.3 further illustrates the situation where the Best Node for execution returned by the Match-Maker is the submission host itself. In that case the job would be submitted to the local scheduler queue and served on FCFS basis.

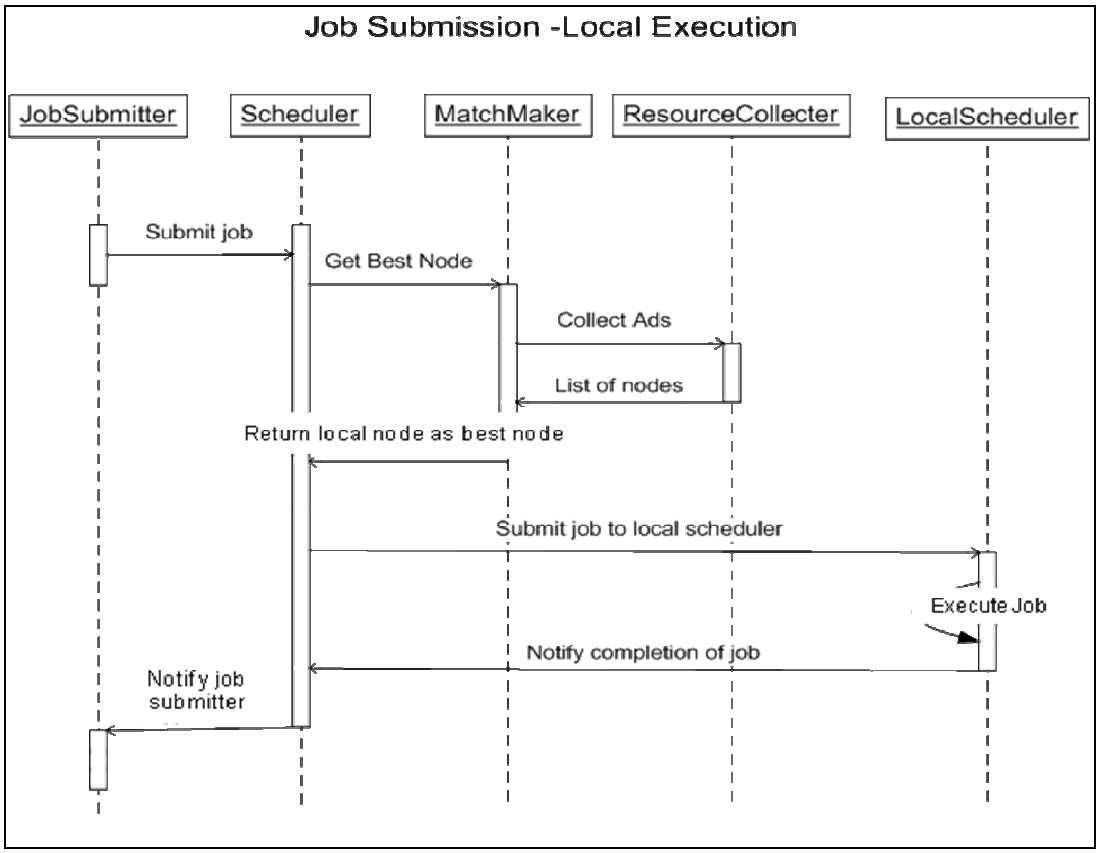

Figure 5.3 Job submission and its local execution

5.6 Remote Execution Sequence Diagram

Figure 5.4 is the sequence diagram that depicts the sequence of events if a Remote Node is returned by the matchmaker for execution. In that case a complex series of events occur.

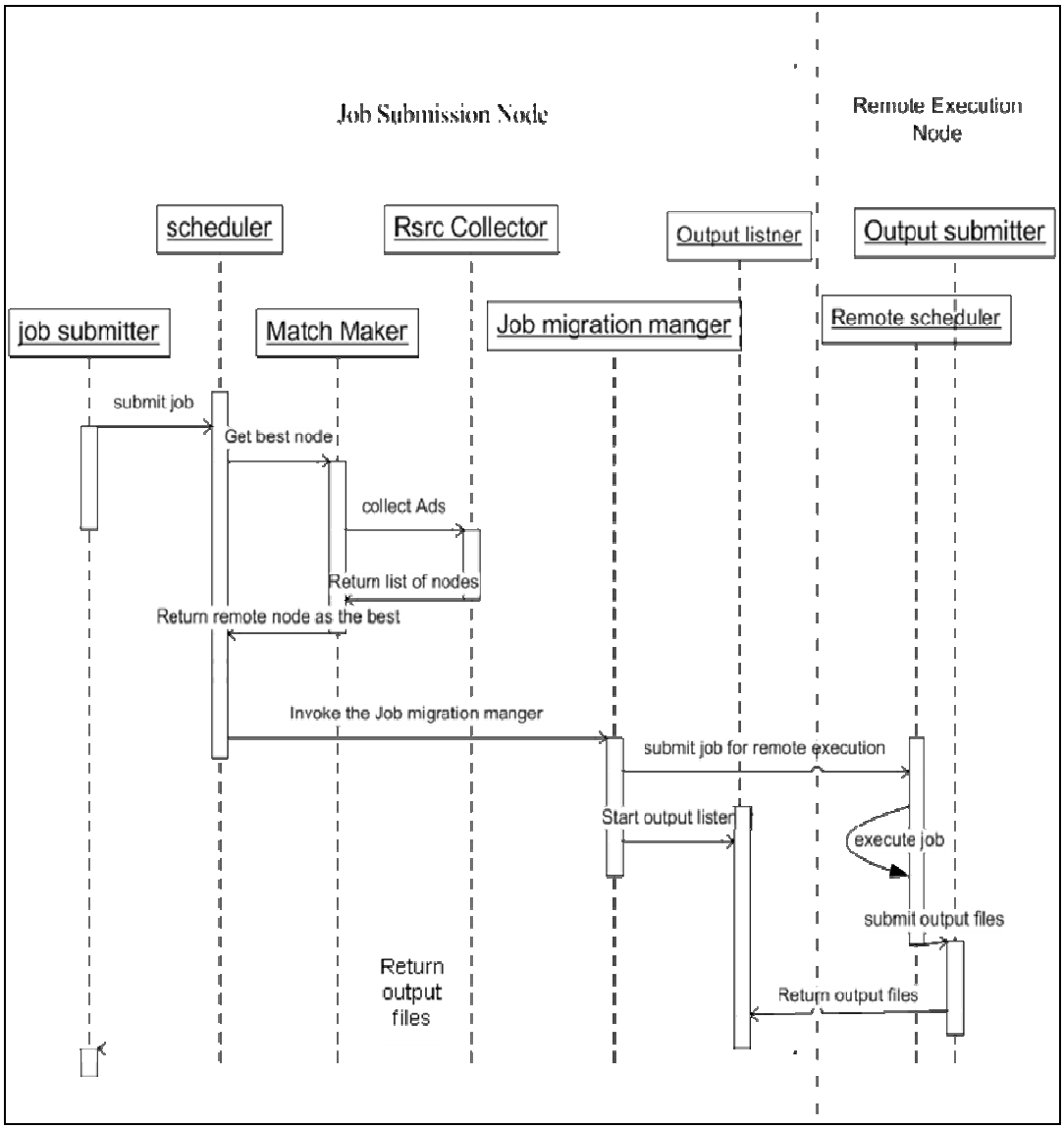

Figure 5.4 Job submission and its remote execution

5.7 DIANA Scheduler in Action

First of all module DaemonMain is executed .It will further create instances of Scheduler, LocalSchedulerQueue, ResourceAdvertiser. After this their threads will be forked.

5.7.1 LocalSchedulerQueue

The object of this class keeps an instance of FCFS Queue. Its job will be to continuously dequeue the job from the queue and send it for execution if any job exists in the queue.

5.7.2 ResourceAdvertiser

It will be continuously listening for any interrupt on the port and when found will send the resource advertisement in the response.

5.7.3 Job Sumission

The job is is submitted by a call to the submit_job () method of the Scheduler. This method will create an instance of the ResourceCollector and will fork its thread. The ResourceCollector thread will send a multicast message over the network to all other connected nodes and will listen to the responses while collecting them in the list. submit_job () will then collect the array containing the resource ads of all the connected nodes. This list will then be passed to the MatchMaker(). It will then perform the matchmaking process based on the list of resources and the resources required by the job. This will return the node that is most suitable for the job to execute on .Now there are actually two cases

5.7.4 Submission node chosen for execution

When the best node is the same node on which the job has been submitted .In this case the job gets submitted in the local queue.

5.7.5 Remote node chosen for execution

The best node is some other node. In this case the job will be given to the JobMigrationManager thread which will; Serialize the Job Contents, make a proxy server of the remote node and call its submitRemoteJob() method, start an OutputListener thread to listen for results.

5.7.6 Job Execution and Output Return

Now when the job is dequeued from the queue, jobExecutioner instance is created and its thread is forked. Now it's the responsibility of the jobExecutioner to execute the job. It will run the job, check its output, error and log files, and fork an OutputSubmitter and thread. OutputSubmitter will open connection with the remote computer (i.e. the computer from which the job had been submitted) and send the output, error and log files (if any) to that remote host. On the other end OutputListener will collect the output and error files.

Chapter 6

TEST RESULTS

6.1 Introduction

In this chapter, results of Iterative Job Submission, Bulk Job Submission and Job Clustering have been illustrated with practical experiments.

6.2 Simulation Setup

Two Clusters consisting of two nodes each were setup. Condor was used as the local node level scheduler on each node. Instance of DIANA metascheduler was running on the central manager of each cluster.

6.3 First Simulation Setup

Jobs were first submitted iteratively to the metascheduler through an automated job submission script.

6.4 Second Simulation Setup

Multiple Jobs were submitted as atomic bulks to the metascheduler which were scheduled and dispatched as atomic bulks

6.5 Third Simulation Setup

Multiple Jobs were submitted as atomic bulks to the metascheduler where these job bulks were clustered into smaller sub-bulks consisting of jobs proportional to the cluster rank.

Table 6.1 Iterative Vs. Bulk Vs. Clustered Approaches

No. of Jobs	Time for Execution(Atomic Iterative Submission)	Time for Execution(Bulk Submission without Job Clustering)	Time for Execution(Bulk Submission with Job Clustering)
10	127	118	74
20	244	228	132
30	368	354	192
40	491	473	246
50	611	591	324
60	727	693	368
70	813	789	408
80	937	892	473
90	1053	1009	536
100	1178	1098	592

Figure 6.1 Effect of Job Clustering over execution speed

Chapter 7

FUTURE WORK

The current implementation of the DIANA Scheduler with job clustering is windows platform compatible. Due to limitations involved in obtaining system information in windows, it is highly intend to make the software Linux compatible to overcome these limitations and base the scheduling on a larger number of involved parameters. The software uses hard coded costs or randomly generated numbers for the time being. It is intended to retrieve the involved parameters in real time and base the scheduling decisions on realistic values.

Discovery service for the software was simulated using multicasting for demonstration purposes in lab only. But this is not a realistic solution for discovery over the internet, since multicast addresses need to be bought exclusively. Multicast addresses are expensive and almost all the possible multicast addresses in the multicast address range have been bought. Another concern while considering multicasting as the solution for discovery over the internet is that ipv4 address range is not capable of fulfilling demands of the exponentially increasing number of internet users. Soon the ipv4 of addresses will be replaced to adjust the large number of internet users and the multicast addresses in use at the time being, will be rendered obsolete. The next solution to address the multicasting problem is still not known. So it is intend to use web services based discovery service implemented by another group of bese-10 with the name ARDIG (Autonomous resource discovery infrastructure for grids). The integration process has already started.

The grid scheduler and the discovery service were developed as services for the project started with the name PhantomOS. The grid Scheduler inherently handles scheduling of batch applications i.e. unattended applications. The final stage of the project will be integration of these services into PhantomOS and running final tests to ensure the services deliver what is required of them.

With the successful integration of the component into PhantomOs, it is intended to write a research paper on “Optimizing throughput through Job clustering” and “two-tier grid scheduler architecture”. Update the PhantomOS website and upload the source at Sourceforge.

BIBLIOGRAPHY

[1] Yan Liu , Iowa yanliu@cs.uiowa.edu, Grid Scheduling, Department of Computer Science University of ,ICE2002, Rome, Italy, 17-19 June 2002.

[2] B.C. Schultheiss, L.C.J. van Rijn and C. Kamphuis , Secure Meta-computing in an Extended Enterprise

[3] Metacomputing. http://www.hpti.com

[4] IBM Solutions Grid for Business Partners: Helping IBM Business Partners to Grid-enable applications for the next phase of e-business on demand.

[5] http://www.cs.cf.ac.uk/Dave/C/node33.html

[6] Douglas Thain, Todd Tannenbaum, and Miron Livny ,Condor and the Grid,.

[7] W. Matthews and L. Cottrell, Achieving High Data Throughput in Research Networks, CHEP 2001, China, 2001.

[8] R. Les Cottrell, Saad Ansari, Parakram Khandpur, Ruchi Gupta, Richard Hughes-Jones, Michael Chen, Larry McIntosh and Frank Leers. 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.

[9] Mathis, Semke, Mahdavi & Ott, The macroscopic behaviour of the TCP congestion avoidance algorithm, Computer Communication Review, 27(3), July 1997.

[10] 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.

[11] 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

[12] 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

[13] A. Ali, A. Anjum, R. McClatchey, F. Khan, M. Thomas, A Multi Interface Grid Discovery System, Grid 2006, Barcelona Spain.

[14] 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, Olso, Norway, June 14, 2005.

[15] A. Anjum, R. McClatchey, H. Stockinger, A. Ali, I. Willers, M. Thomas, M. Sagheer, K. Hasham & O. Alvi. DIANA Scheduling Hierarchies for Optimizing Grid Bulk Job Scheduling. Accepted by 2nd IEEE Int. Conference on e-Science and Grid Computing (e- Science 2006), IEEE Computer Society Press, Amsterdam, The Netherlands, Dec, 2006.

[16] Ashiq Anjum, Richard McClatchey, Arshad Ali and Ian Willers, Member, IEEE, Bulk Scheduling with the DIANA Scheduler, IEEE Transactions on Nuclear Science, vol. 53, issue 6, pp. 3818-3829,2006.