Performance Optimization of Sun N1 Grid Engine Using DRMAA

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
In
Software Engineering

By:

Shilpi Gupta
(80631022)

Under the supervision of:
Dr. (Mrs.) Seema Bawa
Professor & Head, CSED

JUNE 2008

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Performance Optimization of Sun N1 Grid Engine Using DRMAA"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. (Mrs.) Seema Bawa** and refers other researcher's works which are duly listed in the reference section.

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

(Shilpi Gupta)

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

Dr. (Mrs.) Seema Bawa
(Supervisor)
Professor & Head
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned By

(SEEMA BAWA)
Professor & Head
Computer Science & Engineering, Department
Thapar University,
Patiala

(R.K.SHARMA)
Dean (Academic Affairs)
Thapar University,
Patiala.

Acknowledgement

With the deepest sense of gratitude, I would like to express my sincere thanks to my supervisor Dr. (Mrs.) Seema Bawa, Professor & Head, Computer Science and Engineering Department (CSED) for her immense help, stimulating suggestions, constant encouragement and support all throughout this thesis work. She has always been motivating and inspiring me to give my best.

I am equally thankful to Dr. Maninder Singh, Assistant Professor, CSED for sharing his insightful knowledge, providing timely assistance, invaluable time and suggestions which went long way in successful completion of this thesis.

I am also highly indebted to Mr. Inderpreet Chopra, Research Scholar, CSED for his out of the way help, guidance, suggestions and motivation.

I also wish to extend my thanks to Ms. Damandeep Kaur, P.G. Coordinator, CSED for help all throughout. I am thankful to Mr. Balwinder Singh, Lab Supdtt, CSED for providing me with material support. I cannot skip mentioning the help and feedback that I got from Ms. Anju Sharma and Ms. Shashi, Research Scholars, CSED.

The moral support that my friends Neha Sood, Neha Sharma, Gurpreet kaur, Rohit Sharma, Kunal Goel and Maninder Singh have extended in the need of hour is worth appreciable.

I would also like to thank the authors whose work I have consulted and the mailing list members who have reverted to my innumerable queries.

Last but not the least I am eternally grateful to my parents and God for keeping me consistent through all ups and downs.

Shilpi Gupta

(80631022)

Abstract

Grid Computing provides an easy access to distributed resources across the network which may not be available to the user locally. The resources can be in the form of computing clusters, data storage, licensed software etc. An underlying software/hardware known as middleware facilitates the access to these remote resources by the nodes participating in the grid. There are many middlewares available for this purpose like Globus, Alchemi.Net, Condor and Sun N1 Grid Engine.

In general a middleware is responsible for maintaining a central database of the available resources in the grid network and then to schedule the jobs submitted by the users to appropriate resource access sites. Sun N1 Grid Engine (SGE) is one such kind of middleware which accepts jobs submitted by users, schedules them and also performs several other tasks for job management like suspending, aborting, accounting and reporting.

Performance optimization of grid systems is critical but at the same time very much desirable. One important parameter for measuring the performance is scheduling time taken by the middleware, which has been focused in this thesis work. This parameter can be considerably improved if job resource requirement is obtained at the time of job submission from the user and then submit the job to the particular resource category sites, which needs to be already defined by the administrator of the Distributed Resource Management System (DRMS). This would not only reduce the efforts of the middleware but also the scheduling time of the job, finally improving the performance of the middleware.

In this thesis work, optimization of the performance of Sun N1 Grid Engine (SGE) is done by developing a job submitter using Distributed Resource Management Application API (DRMAA) with Java Bindings. Comparision of the scheduling time taken by jobs submitted through the job submitter developed with DRMAA and the GUI interface "QMON" of the SGE is done. The results show the considerable optimization of the performance of SGE on the basis of scheduling time of the job.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Grid Computing

“Grid computing”, much like an electrical and water types of utilities is available "on demand" and is capable of providing an always-available facility, making possible the sharing of computing resources on an extraordinary scale, among an infinite number of geographically distributed groups [1].

Grid Computing enables users to exploit resources that are not available locally, but are available at some remote site that may be at some other geographical place. Grid Computing can be said as more of interconnected computers which are heterogeneous in nature, across different administrative domains and distributed over different geographical areas. This interconnection of the computers and smooth working of the grid is taken care of by a software/hardware known as middleware, which allows access to distributed resources and does scheduling of the jobs along with keeping the whole network secured. There are many middlewares available for this purpose like Globus, Alchemi.Net, Condor and Sun N1 Grid Engine.

A grid is highly heterogeneous, scalable and dynamic in nature;

- Heterogeneity means that there can be different kind of resources, hardware, software, administrative domains, policies and location of the systems.
- Scalability means that a grid can be extended from few computers to a millions of computers across the boundaries, which may degrade its overall performance; hence it must be designed to be extremely fault tolerant.
- Dynamicity means that resources or the systems can anytime participate or leave the grid, therefore the system should be able to extract maximum from the available resource and reschedule the tasks to other resources without affecting the overall progress of the job execution.

Scheduling of the jobs is also a major element of grid computing, which is done by the middleware. Scheduling of jobs can be done on the basis of tickets issued to each job, priority assigned to the job, urgency related to a job, some polices or combination

these. Scheduling of the jobs is done by the middleware; it maintains a central database of the resources that are available across the network. Middleware determines the resource requirement of the jobs then matches them to the available resources and then schedules the jobs to particular remote resource site.

Security is an important issue for grid, a grid being spread across multiple administrative domains having their on security issues and policies, authentication, authorization and confidentiality are the major areas that are to be taken care of.

1.2 Why Grid Computing?

Computers have been proven to be very efficient to solve complex scientific problems. They are used to model and simulate problems of a wide range of domains; for instance medicine, engineering, security control and many more

Although their computational capacity has shown greater capabilities than the human brain to solve such problems, computers are still used less than they could be. One of the most important reasons to this lack of use of computational power is that, despite the relatively powerful computing environment one can have, it is not adapted to such complicated computational purposes.

1.2.1 Exploit unused resources

In most organizations, computing resources are underutilized. Most desktop machines are busy less than 25% of the time (consider that a normal employee works 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage. The easiest use of Grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy; the execution of the task would be delayed. Grid Computing should enable the job in question to be run on an idle machine elsewhere on the network. From this point of view, Grid Computing looks like a great answer to numerous problems without prodigious management requirements. Convenience does not always imply simplicity and grid enabled applications have to meet two prerequisites: first, the application must be executable remotely and with low

additional operating cost; second, the remote machine must meet any special hardware, software, or resource requirements needed to run the application.

Hence, remote computing is not only a matter of sending jobs to distant machines but also choosing the correct machine that has all the required software and hardware. Even though Computing Power (CPU) can be assumed to be the most commonly shared resource [7], it belongs to a large set of underutilized resources that are to be shared on the network such as storage capacity, software, internet connection and the like which also need to be considered.

1.2.2 Increase Computation

To provide users with more computational power, some crucial areas have to be considered [7]:

Hardware Improvement: microprocessor architecture and other resource capabilities of personal computers continuously increase to provide users with additional power. Periodic computational needs: some applications only need computational power once in a while. The system is fully utilized at times and idle the rest of the time.

- Capacity of Idle Machines: machines are often idle and thus their computational power is free to use. The idea would be to use it only during that idle time and leave once the computer is in use.
- Sharing of Computational results: the key to more sharing may be the development of collaboratories [8]: "...centers without walls, in which the nation's researchers can perform their research without regard to geographical location-interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries".

Considering these areas of interest, communication networks in place could act as a medium to provide access to advanced computational capabilities, regardless of the location of resources.

1.3 OGSA: Open Grid Service Architecture

The OGSA architecture [4] is introduced, which is divided in four main layers (Figure 1.1):

Resources (physical or logical), Web services (OGSI), OGSA services and Grid applications.

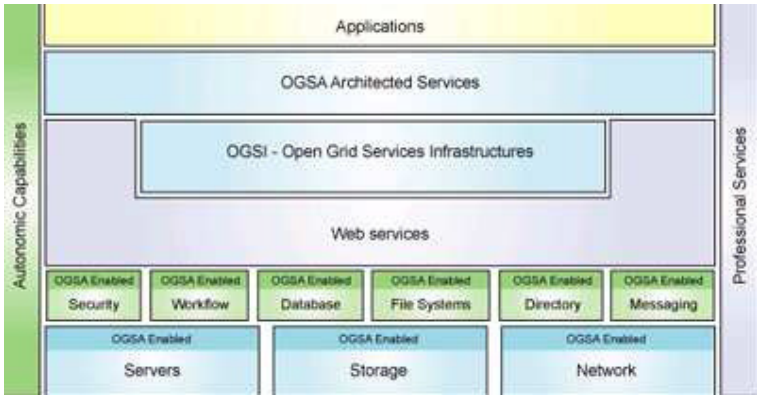


Figure 1.1 OGSA Layered Architecture [3]

1.3.1 Open Grid Service Infrastructure Layer (OGSI)

The Grid infrastructure is built on top of Web services standards, hence leveraging the great effort -in terms of tools and support- that the industry has put into the field. OGSI exploits the mechanisms of Web services like XML and WSDL to specify standard interfaces, behaviors, and interaction for all grid resources. Nevertheless, some key characteristics of Web services standards are missing such as ability to create new services on demand (the Factory pattern); service lifetime management; statefulness; access to state; notification; and service groups. OGSI is based on the concept of a Grid service instance and all services in the OGSA platform must adhere to the conventions specified by OGSI and, therefore, they all are Grid services (Figure 1.2).

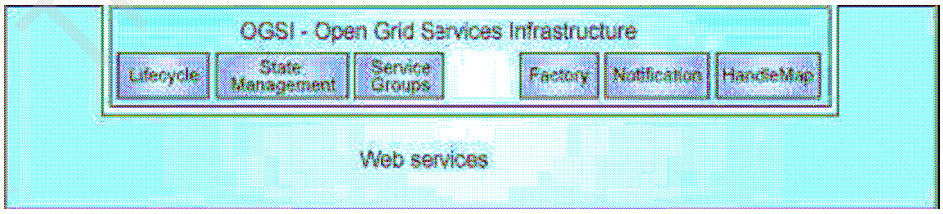


Figure 1.2 OGSI Services [3]

Factory Grid services that implement this interface provide a way to dynamically create and manage new grid service instances. Factories may create temporary instances such as a scheduler creating a service to represent the execution of a particular job, or they may create longer-lived services for frequently used data set.

Life cycle a system that incorporates transient and stateful service instances must be provided with procedures that manage each instance for a specified lifetime. This means that at service destruction, those procedures have to remove all intermediate services that were created. To guarantee those requirements, two standard operations have been defined: Destroy (explicit destruction) and SetTerminationTime (lifetime management). The life cycle mechanism was architected to prevent grid services from consuming resources indefinitely without requiring a large scale distributed "garbage collection" scavenger.

State management Grid services can have state. OGSI specifies a framework for representing this state called Service Data and a mechanism for inspecting or modifying that state named Find/SetServiceData. Further, OGSI requires a minimal amount of state in Service Data Elements that every grid service must support, and requires that all services implement the Find/SetServiceData portType.

Notification Dynamic services have to be able to notify each other about interesting changes as the system runs. This communication of distributed systems is asynchronous. Many interactions between grid services require dynamic monitoring of changing state. In this architecture, service and abstraction interfaces are defined for subscription to (NotificationSource) and delivery of (NotificationSink) these notifications, so that services constructed by the composition of simpler services can deal with notifications (e.g., for errors) in standard ways. The NotificationSource interface is integrated with service data, so that a notification request is expressed as a request for subsequent "push" mode delivery of service data.

HandleMap When factories are used to create a new instance of a grid service, the factory returns the identity of the newly instantiated service. The result of this request is a Grid Service Handle (GSH) and a Grid Service Reference (GSR). A GSR is created with finite lifetime during which it might change while a GSH is guaranteed to reference the grid service indefinitely [5]. The HandleMap interface provides a way to obtain a GSR given a GSH. First, There is a need to identify the HandleMap service that contains the mapping for the precise GSH and then contact this specific HandleMap to get the required GSR.

1.3.2 Physical and Logical Resources Layer

With no doubt, the concept of resources is central to grid computing and thus to OGSA. This common collection can be subdivided into two sets, physical and logical resources which comprise the capabilities of the grid. On one hand, physical resources include servers, storage, and network. On the other hand, logical resources provide additional function by virtualizing and aggregating the resources in the physical layer. From a general point of view, middleware such as file systems, database managers, directories, and workflow managers provide these abstract (logical) services.

1.3.3 OGSA Architected Grid Services Layer

The Web services layer, with its OGSI extensions, provide a base infrastructure for the next layer – architected grid services. It is seen that OGSI was an important step towards the development of a Grid SOA. However, a wide collection of grid services need to be implemented and provided by both middleware software and grid software to ensure application development. This additional layer can be subdivided into four categories (Figure 1.3):

- Domain-Specific Services
- Grid Program Execution Services
- Grid Data Services
- Grid Core Services

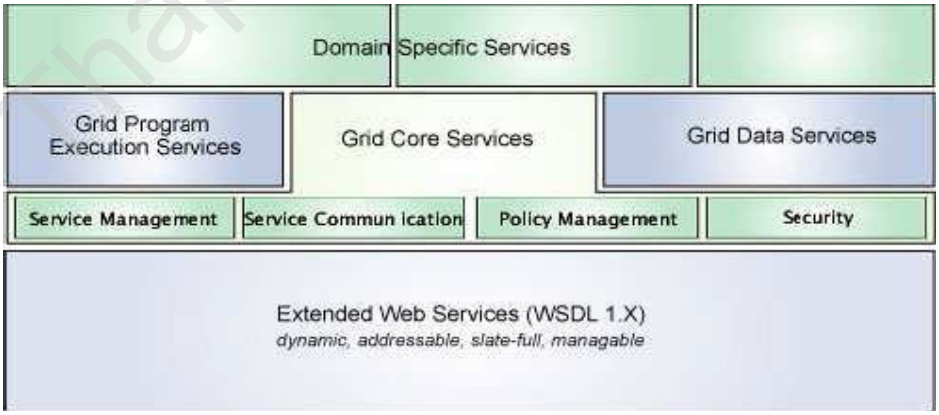


Figure 1.3 Grid Core Services [3]

A closer look at these is taken to understand what the role of this layer, as a whole, is. Grid Core Services Figure 1.3 shows that the grid core services are dividable in four main types of grid services:

- Service management
- Policy management
- Service communication
- Security

These services are to be exploited by most – if not all – higher-level services implemented either in support of program execution or data access, or as domain specific services [3].

Service management is a set of functions that automate and assist maintenance, monitoring and troubleshooting of a service deployed in the Grid. Within this collection, functions for provisioning and deploying the system components as well as collecting and exchanging data (accounting, faults, billing...) about the operation of the grid can be found.

Policy services includes a range of policies managing security, performance, resource allocation and an infrastructure for services (”policy aware”) to make use of policies to drive their operation. It generates a framework that facilitates policy creation, administration and management. Mechanisms for negotiation between service providers and their clients are provided including control of quality and performance.

Service communication comprises support functions for grid services to Communicate with each other. Several communication models are supported to enable interservice communication (message queuing, event notification, logging...).

Security services enable and extend Core Web Services security protocols to provide authentication, authorization, encryption and the like, adapted to a given service. It comprises security models, protocols and technologies for secure communication and data exchange.

Grid Program Execution Services

The two previously defined services are in general applicable to any distributed computing system. In contrast, a Grid program execution class is unique to the grid model of distributed task execution that supports high-performance computing, parallelism, and distributed collaboration. Job scheduling and workload management implemented as part of this class of services are central to Grid computing and the ability to virtualize processing resources.

Grid Data Services

Collection of interfaces supporting data virtualization while providing services that facilitate access to many distributed information such as documents, databases, streaming and files. These services exploit data using placement methods like data replication, data movement assuring QoS across the Grid. Methods for federating multiple disparate, distributed data sources may also provide integration of data stored under differing schemas such as files and relational databases.

1.3.4 Grid Applications Layer

Over time, as a rich set of grid-architected services continues to be developed, new grid applications that use one or more grid architected services will appear. These applications comprise the fourth main layer of the OGSA architecture. The above introduction shows, in a rather detailed style, the structure of OGSA and its components. OGSA is a standard, itself built on standard technologies, that is necessary for realizing distributed heterogeneous computing.

1.4 Grid Applications

A typical example of a Grid application is “weather prediction”. This involves collaboration between several partners: TV stations that produce regular weather news reports, a Satellite Company that regularly provides space images of the earth, a super computing center that rapidly analyses the images and a visualization center that produces visual interpretations of the weather analysis (Figure 1.4). The smooth running of this project for the timely production of regular weather reports crucially depends on appropriate schemas for securely sharing, exchanging, and coordinating information between these partners.

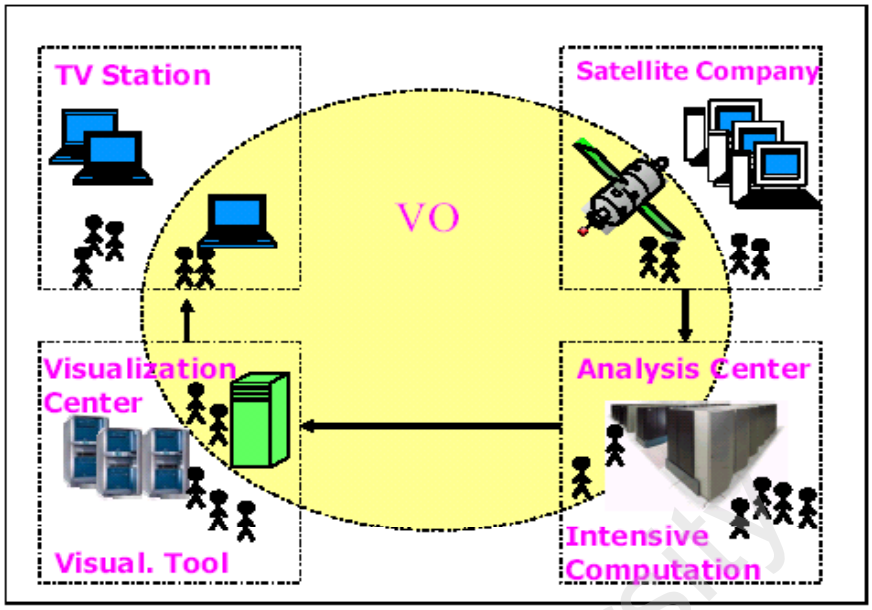


Figure 1.4 Weather prediction using Grid [6]

The power of Grid is particularly useful in arenas involved in intensive processing such as life science research, financial modeling, industrial design, and graphics rendering [6]. Many governments have recently initiated special programmes to support the Grid: The benefits of having partnerships between institutions to achieve ambitious projects have been well recognized.

1.5 Components of Grid

Grid computing can be divided into six major components [3]. A detailed description of each of those is given in the subsequent sections and thus only a brief introduction of the reasons why they are considered as major is given here.

1.5.1 User Interface

Accessing information on the grid is fairly important, and the user interface [3] component handles this task for the user. It often comes in one of two ways:

- An interface provided by an application that the user is running.
- An interface provided by the grid administrator, much like a Web portal that provides access to the applications and resources available on the grid in a single virtual space.

The use of a user interface as a portal is also important because it can help users to learn how to query the grid.

1.5.2 Security

Computers on a grid are networked and running applications; they can also be handling sensitive or extremely valuable data, so the security component of grid computing is of crucial concern. This component includes elements such as encryption, authentication, and authorization.

1.5.3 Workload Management

Applications that a user wants to run on a grid must be aware of the resources that are available; this is where a workload management service comes in handy. An application can communicate with the workload manager to discover the available resources and their status.

1.5.4 Scheduler

A scheduler is needed to locate the computers (resources) on which to run an application, and to assign the jobs required i.e. scheduler map the heterogeneous resources with application's tasks. This can be as simple as taking the next available resource, but often this task involves prioritizing job queues, managing the load, finding idle machines.

1.5.5 Data Management

If an application is running on a system that doesn't hold the data the application needs, a secure, reliable data management facility takes care of moving that data to the right place across various machines, encountering various protocols.

1.5.6 Resource Management

To handle such core tasks as launching jobs with specific resources, monitoring the status of those jobs, and retrieving results, a resource management facility is necessary. It's important to remember that grid computing doesn't operate in a vacuum–just the opposite. It potentially involves every protocol and computer technology in operation today.

1.6 Thesis Framework

This section discusses the framework of this thesis. This thesis is organized as follows:

Chapter 2 deals with the literature review, it starts with the history of grid computing, then moves on to describing different middlewares available for grid systems. Then it gives an overview of different APIs available for grid. The last section of this chapter describes how grid API DRMAA and SGE work together.

Chapter 3 is the problem definition; this chapter gives a view of the limitations of the existing system. The next section describes the solution proposed to overcome these limitations and improve the performance of the system. The methodology used to implement the solution is discussed in the last section of this chapter.

Chapter 4 describes how the problem defined in chapter 3 can be solved and how the implementation of the solution is done. The first section of the chapter describes how a grid environment using SGE is set up. The next section discusses the use of DRMAA with java bindings. The later section gives the details of development of a job submitter using DRMAA as a solution to the problem defined in previous chapter. The jobs used for experiments are discussed in the next section. Then the job submission through QMON and DRMAA is described.

Chapter 5 gives the experimental results, analysis and comparison of the job submission through QMON and DRMAA. The results and analysis of the experiments are presented in the form of graphs.

Chapter 6 presents the conclusion of the thesis and highlights future research directions based on the results obtained.

Chapter 2
Literature Review

2.1 Evolution of Grid Computing

The term "Grid" was coined in the mid1990s to denote a proposed distributed computing infrastructure for advanced science and engineering. The concept of Computational Grid has been inspired by the 'electric power Grid', in which a user could obtain electric power from any power station present on the electric Grid irrespective of its location, in an easy and reliable manner. When additional electricity is required, there is need of just plugging into a power Grid to access additional electricity on demand, similarly for computational resources plug into a Computational Grid to access additional computing power on demand using most economical resource [11].

Technology	Year
Networked Operating Systems	1979-81
Distributed operating systems	1988-91
Heterogeneous computing	1993-94
Parallel distributed computing	1995-96
The Grid	1998

Table 1.1 Historical Background of Grid [12]

The theory behind "Grid Computing." is not to buy more resources but to borrow the power of the computational resources you need from where it's not being used". Many of the basic ideas behind the Grid have been around in one form or other throughout the history of computing. One of the "novel" ideas of the Grid is sharing computing power.

There is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented; it is reinvented in a much more powerful form, because computer processors, memories and networks improve at exponential rates [13]. Because of the huge improvements of the underlying hardware (typically

more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor.

2.2 Different Middlewares for Grid

Grid is more easily stated as consisting of shared heterogeneous computing and data resources networked across administrative boundaries. Thus, a grid can be thought of as both an access method and a platform, with grid middleware being the critical software that enables grid operation and ease-of-use. For a grid to function effectively, it is assumed that

- Hardware and software exists on each resource to support participation in a grid and,
- Agreements and policies exist among grid participants to support and define resource sharing [2].

Availability of resources in a grid environment being highly dynamic in nature, it's the middleware or job resource manager who is responsible of providing resources to a particular job and scheduling it to different distributed resources across the grid network. There are different middlewares available for this purpose, some of which are described below:

2.2.1 Alchemi.Net

There is a distinct lack of service oriented architecture-based grid computing software in Microsoft Windows based grid computing infrastructure. To overcome this limitation, Windows-based grid computing framework called Alchemi [20] [21] [22] is implemented on the Microsoft .NET Platform.

The Microsoft .NET Framework provides a powerful toolset that can be leveraged for all of these, in particular support for remote execution (via .NET Remoting and web services), multithreading, security, asynchronous programming, disconnected data access, managed execution and cross-language development, making it an ideal platform for grid computing middleware.

The Alchemi grid computing framework was conceived with the aim of making grid construction and development of grid software as easy as possible without sacrificing flexibility, scalability, reliability and extensibility. The key features supported by Alchemi are:

- Internet-based clustering of desktop computers without a shared file system.
- Federation of clusters to create hierarchical, cooperative grids.
- Dedicated or non-dedicated (voluntary) execution by clusters and individual nodes.
- Object-oriented grid thread programming model (fine-grained abstraction).
- Web services interface supporting a grid job model (coarse-grained abstraction) for crossplatform interoperability e.g. for creating a global and cross-platform grid environment via a custom resource broker component.

The architecture of Alchemi follows the master-worker parallel programming paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. This smallest unit of parallel execution is a grid thread, which is conceptually and programmatically similar to a thread object (in the objectoriented sense) that wraps a "normal" multitasking operating system thread. A grid application is defined simply as an application that is to be executed on a grid and that consists of a number of grid threads. Grid applications and grid threads are exposed to the grid application developer via the object-oriented Alchemi .NET API.

The following are the components of Alchemi:

(A) Manager: The Manager manages the execution of grid applications and provides services associated with managing thread execution. The Executors register themselves with the Manager which in turn keeps track of their availability. Threads received from the Owner are placed in a pool and scheduled to be executed on the various available Executors. A priority for each thread can be explicitly specified when it is created within the Owner, but is assigned the highest priority by default if none is specified. Threads are scheduled on a Priority and First Come First Served (FCFS) basis, in that order. The Executors return completed threads to the Manager which are subsequently passed on or collected by the respective Owner [11].

(B) Executor: The Executor accepts threads from the Manager and executes them. An Executor can be configured to be dedicated, meaning the resource is centrally managed by the Manager, or non-dedicated, meaning that the resource is managed on a volunteer basis via a screen saver or by the user [11].

(C) Owner: Grid applications created using the Alchemi API are executed on the Owner component. The Owner provides an interface with respect to grid applications between the application developer and the grid. Hence it "owns" the application and provides services associated with the ownership of an application and its constituent threads. The Owner submits threads to the Manager and collects completed threads on behalf of the application developer via the Alchemi API [20].

(D) Cross-Platform Manager: The Cross-Platform Manager is an optional subcomponent of the Manager, it is a generic web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of platform independent grid jobs (as opposed to grid applications utilizing the Alchemi grid thread model). Jobs submitted to the Cross-Platform Manager are translated into a form that is accepted by the Manager (i.e. grid threads), which are then scheduled and executed as normal in the fashion described above. Thus, in addition to supporting the gridenabling of existing applications, the Cross-Platform Manager enables other grid middleware to interoperate with and leverage Alchemi on any platform that supports web services (e.g. GridBus Grid Service Broker) [11].

2.2.2 Condor: Cycle Stealing Technology for High Throughput Computing

Condor [22] [23] is a high-throughput computing environment developed at the University of Wisconsin at Madison, USA. It can manage a large collection of computers such as PCs, workstations, and clusters that are owned by different individuals. Although it is popularly known for harnessing idle computers CPU cycles (cycle stealing), it can be configured to share resources. The Condor environment follows a layered architecture and offers powerful and flexible resource management services for sequential and parallel applications. The Condor system pays special attention to the computer owner's rights and allocates their resources to the Condor pool as per the usages conditions defined by resource owners. Through its unique remote system call capabilities, Condor preserves the job's originating machine

environment on the execution machine, even if the originating and execution machines do not share a common file system and/or user ID scheme. Condor jobs with a single process are automatically check-pointed and migrated between workstations as needed to ensure eventual completion. The Condor has been extended to support submission of jobs to resources Grid-enabled using Globus services. Condor can have multiple Condor pools and each pool follows a flat machine organization. The Condor collector, which provides the resource information store, listens for advertisements of resource availability. A Condor resource agent runs on each machine periodically advertising its services to the collector. Customer agents advertise their requests for resources to the collector. The Condor matchmaker queries the collector for resource discovery that it uses to determine compatible resource requests and offers.

The agents are then notified of their compatibility. The compatible agents then contact each other directly and, if they are satisfied, then the customer agent initiates computation on the resource. Resource requests and offers are described in the Condor classified advertisement (ClassAd) language. ClassAds use a semi-structured data model for resource description. Thus, no specific schema is required by the matchmaker allowing it to work naturally in a heterogeneous environment. The ClassAd language includes a query language as part of the data model, allowing advertising agents to specify their compatibility by including constraints in their resource offers and requests. The matchmaker performs scheduling in a Condor pool. The matchmaker is responsible for initiating contact between compatible agents. Customer agents may advertise resource requests to multiple pools with a mechanism called flocking; allowing a computation to utilize resources distributed across different Condor pools. The scheduler is centralized [24].

2.2.3 Globus

Globus provides a software infrastructure that enables applications to view distributed heterogeneous computing resources as a single virtual machine. The Globus project is an American multi-institutional research effort that seeks to enable the construction of computational Grids. A central element of the Globus system is the Globus Toolkit [25], which defines the basic services and capabilities required for constructing computational Grids. The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, data management,

resource reservation, and communications. The toolkit provides a bag of services from which developers of specific tools or applications can select, to meet their own particular needs.

Globus is constructed as a layered architecture in which higher-level services can be developed using the lower level core services. Its emphasis is on the hierarchical integration of Grid components and their services. This feature encourages the usages of one or more lower level services in developing higher-level services. Resource and status information is provided via an LDAP-based network directory called Metacomputing Directory Services (MDS). MDS consists of two components, Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS). GRIS implements a uniform interface for querying resource providers on a Grid for their current configuration, capabilities, and status. GIIS pulls the information from multiple GRIS services and integrates it into a single coherent resource information database. The resource information providers use a push protocol to update GRIS. Thus MDS follows both push and pull protocols for resource dissemination. Higher-level tools such as resource brokers can perform resource discovery by querying MDS using LDAP protocols. The MDS namespace is organized hierarchically in the form of a tree structure. Globus offers QoS in the form of resource reservation [24].

2.3.4 Sun N1 Grid Engine

Sun's Grid Engine is a Distributed Resource Management tool; it provides load management across heterogeneous, distributed computing environments. Sun made the source available under the Grid Engine project [26]. The Grid Engine project is an open source community effort, sponsored by Sun, to further distributed computing applications and services. The open source version of the software is known as Grid Engine. Grid Engine is generally installed across a cluster of machines using a shared file system. It can be configured to work across disparate file systems. Users submit jobs to Grid Engine and Grid Engine manages the allocation of jobs to machines within the cluster. This ensures the resources are used more productively therefore increasing availability. There are various components (Figure 2.1) that make up a Sun Grid Engine cluster.

(A) Hosts: There are different types of host in a Grid Engine cluster:

Master Host: The master host acts as the core of this centralized system. The daemons qmaster and schedd run on the master host. There is only one master host.

Execution Host: An execution host is a node within the cluster that is capable of executing jobs. Each execution host runs the execution daemon execd.

Administration Host: Administration hosts are nodes that are given permission to alter the configuration of the Grid Engine cluster. By default the master host is also an administrative host.

Submit Host: Submit hosts are nodes within the cluster that users are allowed to submit jobs from to the Grid Engine. By default the master host is also a submit host.

Shadow Host: This node monitors the master host. If the master host fails, the shadow host takes over. If there is more than one shadow host an election protocol ensures only one takes over as master.

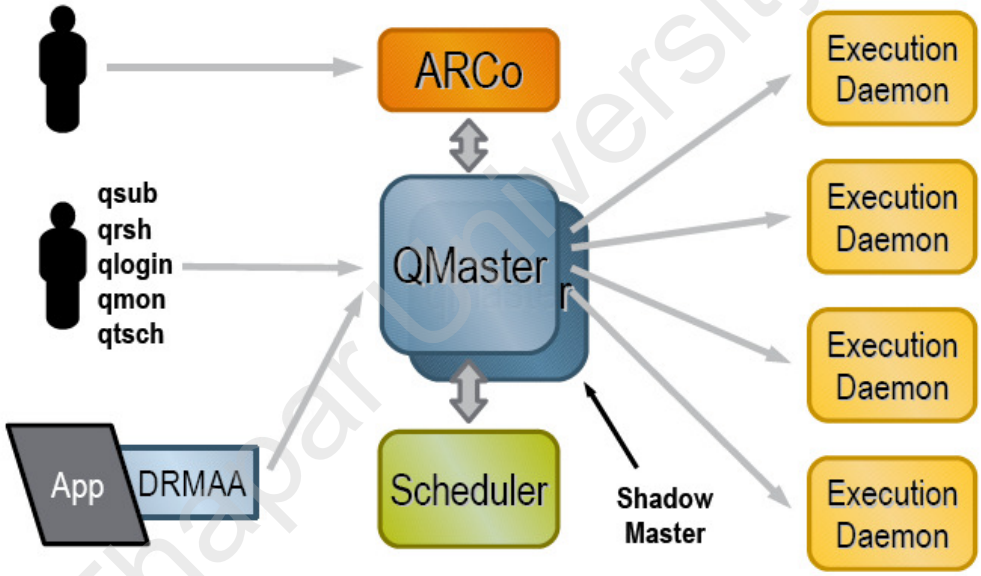


Figure 2.1 N1 Grid Engine Components [28]

(B) Daemons: Daemons are processes that run background on nodes of the Grid Engine cluster.

Master Daemon: The qmaster daemon is central to the Grid Engine. It handles all requests to the Grid Engine Database and coordinates with the scheduler daemon and execution daemons.

Scheduler Daemon: The schedd is responsible for retrieving the current state of the cluster from qmaster and deciding which queue a job should be assigned to.

Execution Daemon: The execd is responsible for running jobs for the queue(s) configured for the execution host it runs on. It receives requests from qmaster, such as to run a job, and sends job reports and load reports back to the qmaster.

Shepherd Daemon: An individual shepherd is started by the execd as the parent process of each job. The shepherd starts the job and collects resource usages statistics once the job finishes.

Communication Daemon: The commd is used to separate communication from processing. All messages between daemons and/or hosts are sent via the commd.

Shadow Daemon: The shadowd runs on a shadow host. This daemon monitors the qmaster daemon. If the shadowd detects the qmaster has failed all running shadowds decide which is to take over as the new qmaster.

(C) Queues: Queue represents a class of jobs allowed to execute concurrently on an execution host. Jobs do not wait in a queue, once dispatched to a queue the job is associated with that queue and runs on that execution host. If the queue is suspended then so are any running jobs. The only place jobs wait are in the scheduler's job pending list.

Queues are configured with various attributes, for example the number of processors. Job requirements have to fit within queue attributes for that queue to be considered as a suitable place for the job to run. If there is more than one suitable queue the scheduler will base its decision in a configurable manner: it can choose least loaded queue or based on queue sequence number [26].

(D) Complexes: Complexes are used to provide all the useful information about attributes of resources used within Grid Engine. They are also used as the framework for the Grid Engine's Consumable Resource facility, where attributes have an associated capacity that is monitored. They are used to describe queue, host and the global cluster attributes. The description includes name, shortcut, type, value, relation, requestable, consumable and default.

2.3 APIs for Grid

In the Grid world, there are different systems for several tasks, like number of environments for jobs scheduling, file transfers, resource discovering etc. These tasks complicate applications and make porting really difficult.

To address these challenges and in particular to find a solution to the universal, apparently intractable problem of successfully Grid-enabling applications, several applications groups expressed the desire for a simple programmatic interface that is widely-adopted, usable and available. The goal of such an interface would be to provide developers with a simple, uniform, and standard programmatic interface with which to develop distributed applications. This interface is known as Grid Application Programming Interface (Grid API). There are several APIs available for grid applications as discussed in following sections.

2.3.1 GridRPC

A commonly used way of network programming is Remote Procedure Call. There was a need to port this concept to the Grid environment also. Grids are special in the context of RPC programming [14] because of:

- Dynamic resource discovery and scheduling,
- Scientific datatypes (large matrices),
- Big variety of calls complexity (from a second to days or weeks).

Hence, GridRPC [14, 15] offers API for function calls, execution control and synchronization. It hides service discovery behind simple API call, i.e. obtaining function handle.

2.3.2 SAGA

The Simple API for Grid Applications (SAGA) [16], has the potential to become an important specification, due to the current problems for application developers, which revolves around the rapid rate of change in middleware standards and APIs, its complexity, and the fact that different middleware exists on different grid systems. SAGA aims to be to the grid application developer what MPI has been to the developer of parallel application. In SAGA object-oriented approach is used and there are different packages responsible for different areas of grid subsystems (session

handling and security, jobs and tasks, data management, steering and monitoring, task dependencies) [17].

2.3.3 DRMAA

A key component of the grid is a distributed resource management system: software that queues, dispatches, and controls jobs. The Distributed Resource Management Application API (DRMAA) working group [18] has released the DRMAA specification, which offers a standardized API for application integration with C, Java, and Perl bindings. DRMAA can be used to interact with batch/job managements systems, local schedulers, queuing systems, and workload management systems. DRMAA has been implemented in a number of DRM systems, including Sun's Grid Engine, Condor, PBS/Torque, Gridway, gLite, and UNICORE.

2.4 Sun N1 Grid Engine and DRMAA

Enterprises use DRM (Distributed Resource Manager) software, such as Sun N1 Grid Engine, to radically increase throughput in their data centers. The critical factor in making DRM grid software productive is seamlessly integrating it into the existing IT infrastructure. A good DRM application integration will allow the users to continue working as usual, keeping the grid software totally transparent to them. The only changes the users should notice are that results are delivered faster, accelerating everyday operations in the users' environments and dramatically improving productivity [27].

DRMAA is a software standard developed in the GGF (Global Grid Forum) by representatives from among the world's leading DRM vendors to make interaction with DRM systems more uniform, in an effort to generally promote integrated job processing. An application integrated with the standard DRMAA interface can be run on any DRM software that has adopted the DRMAA standard.

Anyone who is planning a seamless integration with N1 Grid Engine should consider using DRMAA for their system:

- End Users who must control the job workflow in their work
- Enterprise system integrators (internal or external) who support the software infrastructure for N1 Grid Engine

- ISV's (Independent Software Vendors) of applications that are typically run through
- DRM systems
- N1 Grid Engine OEMs

The DRMAA standard substantially increases the value of such integrations compared to using DRM-specific interfaces, such as qsub, qstat, qdel, etc. in the case of N1 Grid Engine [27].

Chapter 3
Problem Formulation

3.1 Limitations of the existing system

Most of the users that use a grid system, submit jobs to the system through a user interface that has been provided by the administrator of the Distributed Resource Management System (DRMS). Often users submit the job and leave the major tasks of determining the resource requirement and scheduling of job to the middleware.

The SGE middleware system features a graphical user interface (GUI) command tool, the QMON Main Control window. The QMON Main Control Window (Figure 3.1) enables users to perform most grid engine system functions, including submitting jobs, controlling jobs, and gathering important information [27].

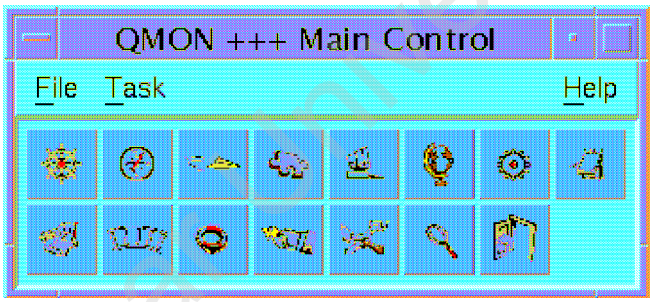


Figure 3.1 QMON Main Control Window [27]

The limitations observed while using this graphical interface are:

- Administrator of the DRMS cannot determine the resource requirement of the jobs before its submission to the system.
- There is no provision of submitting the jobs to a particular resource category site.
- Administrator of the DRMS is unable to define different methods of overriding policies specific to different resource sites for the same job.
- Due to more command line interactions while job submission through QMON, the performance of SGE based on scheduling time of job is reduced.

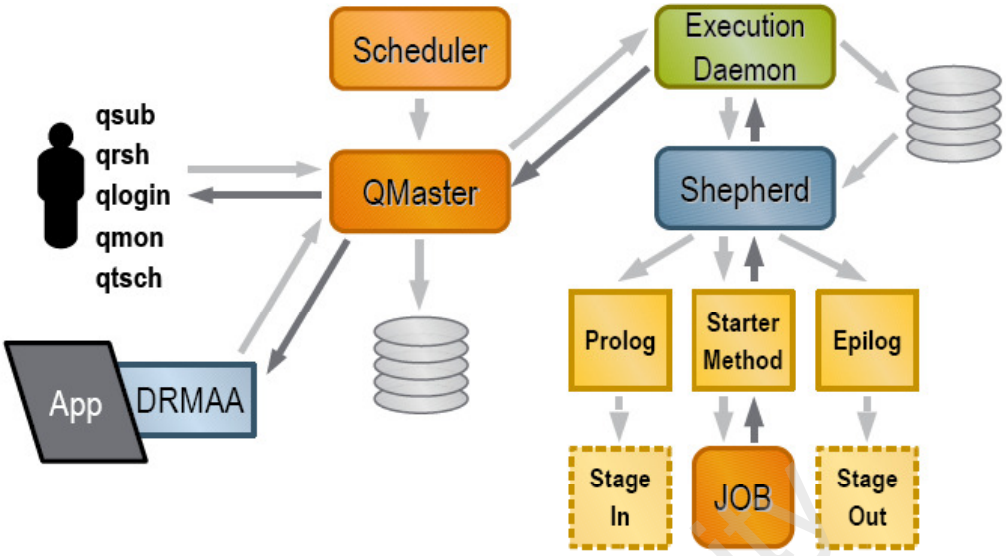


Figure 3.2 Life Cycle of a Job in SGE [28]

3.2 The Proposed Solution

The problems or the limitations defined in the above section of this chapter are proposed to be solved by developing a new Job Submitter interface using an API for grid called DRMAA (Distributed Resource Management Application API) with Java Bindings.

The proposed solution aims at solving the problems defined in the above section along with optimizing the performance of the SGE on the basis of time taken to schedule the jobs. The new job submitter that has been proposed will take the resource requirements of the job from the user at the time of submission of the job and would allow the user to submit the job to particular resource requirement sites and override the site policies using the job categories that need to be already defined by the administrator of the DRMS.

As the new Job Submitter is being developed using DRMAA, it shall increase the performance of SGE because DRMAA provides a fundamental set of operations allowing programmatic access to capabilities common to all DRM systems for integrated environments. Due to session-based programming model provided by DRMAA, interactions with DRM systems are comparatively faster than interactions

via the command line. For example, Due to lower job submission overhead, job submission rates can easily be doubled just by using DRMAA [27].

3.3 Methodology Used

To implement the proposed solution of the problem that is being taken care of in this thesis work, the following methodology is used:

I. A Grid Environment using Sun N1 Grid Engine (SGE) as middleware is set up for performing desired experiments.
II. The study of how to use DRMAA with SGE is undertaken
III. A new Job Submitter using DRMAA with Java Bindings is developed for the purpose of overcoming the limitations and improving the performance of SGE.
IV. Three job scripts are used for performing experiments and show the desired improvement in the performance of the SGE. Two example job scripts available with SGE package are used and one job script is developed for the same purpose.
V. The job submission experiments are performed and scheduling time of the jobs is noted down of jobs submitted through QMON and DRMAA Job Submitter.
VI. The comparative analysis of the results obtained from the experiments is done.
VII. Final results of achieving the proposed solution and optimization of performance of the SGE are shown through graphs.

The implementation details of the above methodology are given in the next chapter.

Chapter 4
Implementation Details

This chapter discusses the implementation details of the solution proposed for the problems defined in the previous chapter. First section of this chapter describes the setting up of grid environment. The next section gives a brief view of using DRMAA with java bindings as this has been used for developing the proposed job submitter

Third section of this chapter discusses that how the job submitter has been developed for a grid environment using SGE as middleware with the help of DRMA API Java Bindings, its features and parameters. The next section discusses the jobs that are taken for performing the experiments. Lastly, this chapter gives the details about how jobs are submitted through QMON and DRMAA job submitter and outputs that are obtained.

4.1 Setup of Grid Environment using Sun N1GE

There are three V20Z Sun Fire Rack Servers, which are used to set up a grid environment and build a cluster of Sun N1GE. On these three servers Solaris 10 operating system is installed and hostnames as tietgrid8, tietgrid01 and tietgrid02 are given to them. The prerequisite of making a cluster of systems is that they should be interconnected through LAN and be able to communicate with each other. The systems are already connected via LAN. SGE version 6.1 update 3 is installed on them.

The master host is installed on the system with hostname tietgrid8; it acts as administration host, submission host, and also as execution host. Then execution host is installed on servers with hostnames tietgrid01 and tietgrid02 respectively. They act as submission host, execution host and administration host, a node has to be an administration host before it can be an execution host.

The detailed steps of installation of SGE are given in the Appendix - A at the end of this thesis.

4.2 Using DRMAA with Java Bindings

To use the DRMAA with Java language binding implementation included with Sun N1 Grid Engine 6.1, location of important files should be known. The most important file is the DRMAA JAR file *sge-root*/lib/drmaa.jar. To compile a DRMAA application, DRMAA JAR file must be included in the CLASSPATH. The DRMAA classes are documented in the DRMAA Javadoc, located in the *sge-root*/doc/javadocs directory. To run the application, a DRMAA shared library, *sge-root*/lib/*arch*/libdrmaa.so, is also needed which provides the required native routines. To use the DRMAA classes in the application, classes of applications should import the DRMAA classes or packages. Only the classes in the org.ggf.drmaa package will be used. These packages can be imported individually or using a wildcard package import.

The DRMAA shared library, which is used by default, supports version 1.0 of the DRMAA Java Language Binding Specification.

4.3 Development of Job Submitter using DRMAA

A Job Submitter using DRMAA with Java Bindings is developed, for development NetBeans IDE 5.5 is used. Detailed steps of using DRMAA with NetBeans IDE 5.5 are given in Appendix - B at the end of this thesis.

For this Job Submitter using DRMAA, few job categories are defined for a job. Site administrators may create a job category suitable for an application to be dispatched by the DRMS; the associated category name could be specified as a job submission attribute. The DRMAA implementation may then use the category name to manage site-specific resource and functional requirements of jobs in the category. An example can illustrate this idea:

At site A, an application X is used in a heterogeneous clustered environment that is managed by a DRMS. Since application X is available only at a subset of these machines, the administrator wants to set up the DRMS so that the end users may be able to submit jobs to this subset of machines only.

At site B, the same application is used in a homogeneous clustered environment with an application X supported at all machines managed by the DRMS. However, since X

jobs do compete with applications Y sharing the same resources and X applications are to be treated with higher priority than Y jobs, end users need to specify some priority in their submit command line for raising the dispatch priority.

An integration based on categories will allow submitting X jobs through the DRMAA interface in compliance with the policies of both sites A and B without the need to know about these policies. This can be done by specifying "X" as the category used for X jobs submitted through the DRMAA interface and by mentioning this in the "DRMS integration" section of the X jobs software documentation.

The administrators at site A and site B will read the documentation or installation instructions about the "X" DRMAA category. The documentation of their DRMS contains directions about the category support of their DRMAA interface implementation. From this documentation they learn how to configure their DRMS in a way that how "X" jobs can be submitted to particular subset of machines at site A and set priority for “X” jobs at site B for those jobs.

The above idea is implemented by defining job categories as category1, category2, and category3 in the DRMAA job submitter. The administrator of the DRMS according to his needs may modify the definition of these job categories. Defining these job categories beforehand would reduce the effort of the middleware in determining the resource requirements and scheduling of the job, eventually improving the performance of the SGE. It is also a boon for a novice user who may not know how to get his jobs submitted to a particular resource category site.

Working of DRMAA Job Submitter

I. The job category along with job path is taken as input from the user.
II. A new job session template is created for every job.
III. The definition of the job categories is specified in the hashmap table of the program code.
IV. On the basis of the input taken, hashmap is resolved to get corresponding definition argument for submitting the job to SGE.
V. This argument is then passed to nativespecification() function provided in DRMAA.
VI. The job path is passed to jobtemplate’s remote command function.

VII. The job is run through a runjob() function of the session.

VIII. A message of successful submission of the job is shown.

IX. After collecting the parameters related to the job and checking its finish status the corresponding status message and parameters related to the job are shown and session jobtemplate is deleted.

4.4 Jobs for Experiments

Some sample job scripts are required for performing the experiments. In this thesis work, three job scripts that are coded in shell script are taken for the purpose of experiments. These jobs are for serial execution and not for parallel execution. The description of jobs taken for experiments is as follows:

Sleeper.sh is a job script available in example scripts bundled with SGE, in this job script a new thread is created and made to sleep for 1000ms.

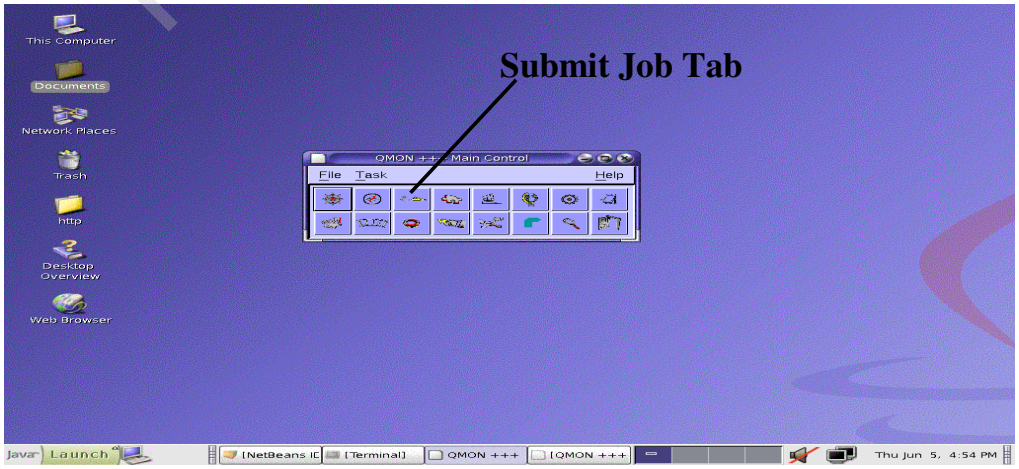
Simple.sh is again a job script available in example scripts bundled with SGE, this job script prints the current date and time of the system twice as an output.

Prime.sh is a job script that has been developed for experiments in this thesis work; this job script generates all prime numbers in the range 1 to 1000.

4.5 Job Submission through QMON

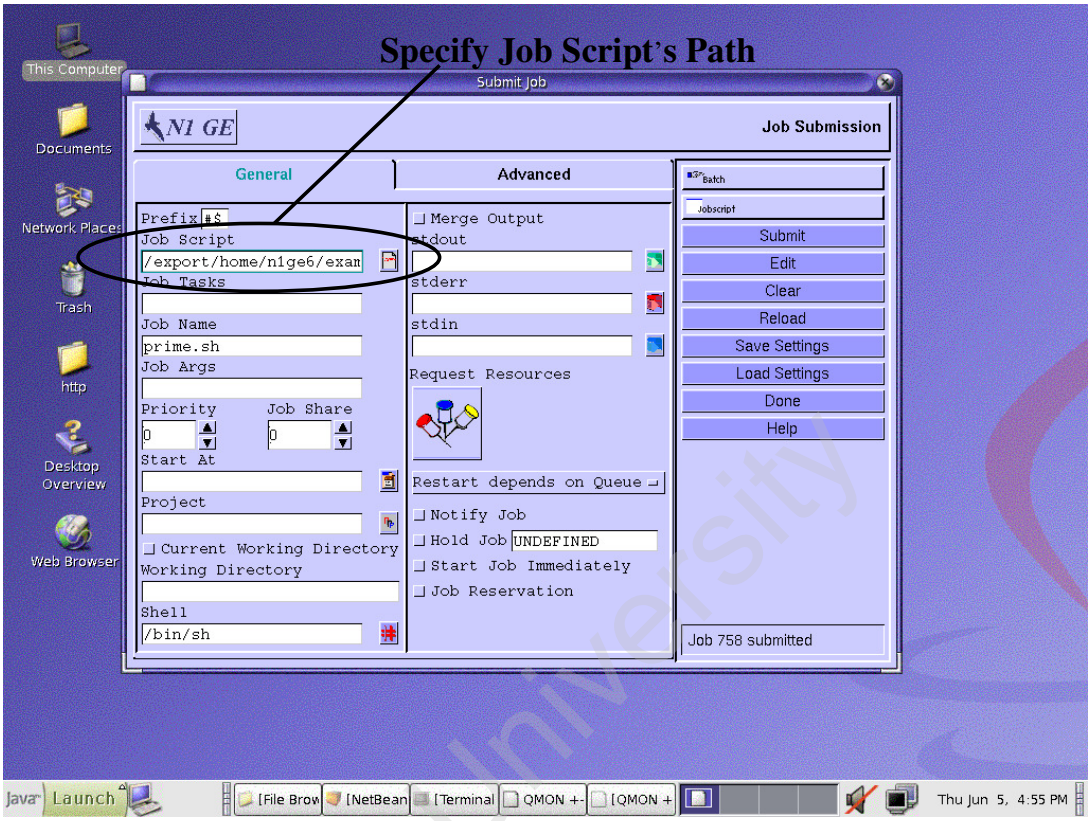
This section shows how the jobs are submitted through QMON to the DRMS.

1. The Screenshot 4.1 shows the job submission tab on the QMON control window.



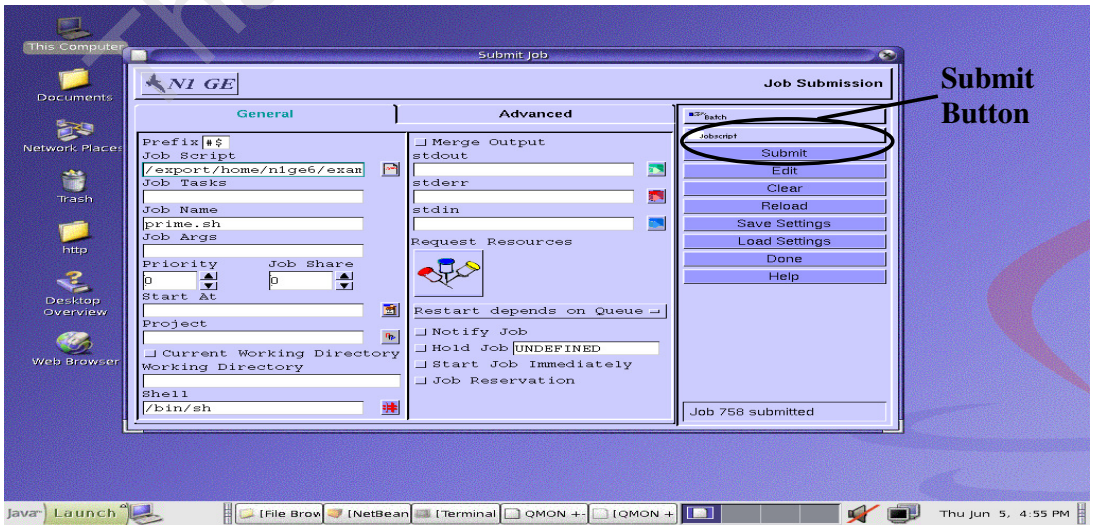
Screenshot 4.1 Submit Job Tab on QMON Control Window

2. The Screenshot 4.2 shows where the job path has to be specified.



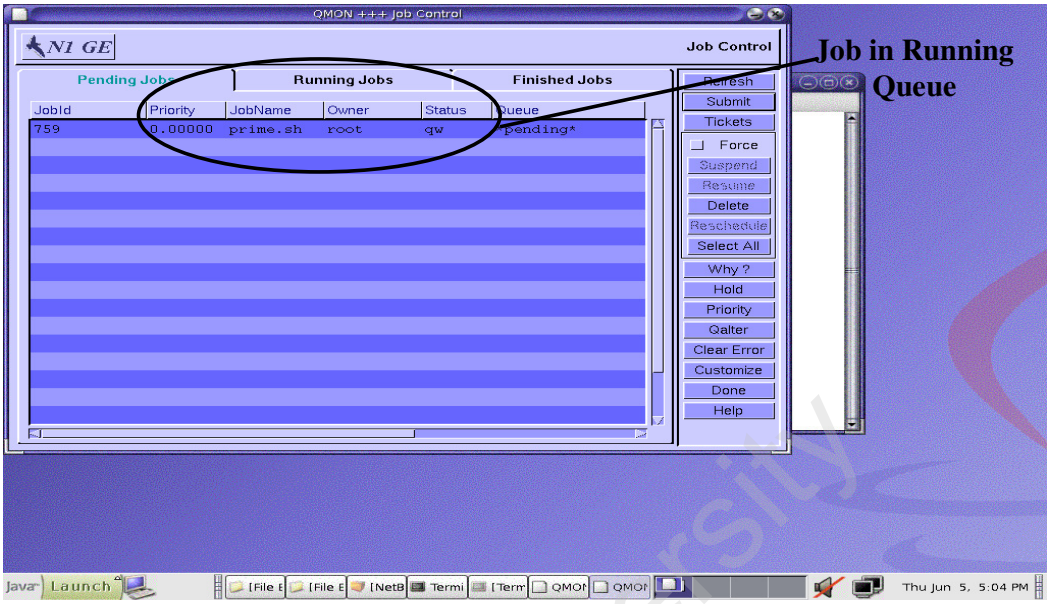
Screenshot 4.2 Submit Job window to specify job script's path

3. Then click submit button to submit the job.



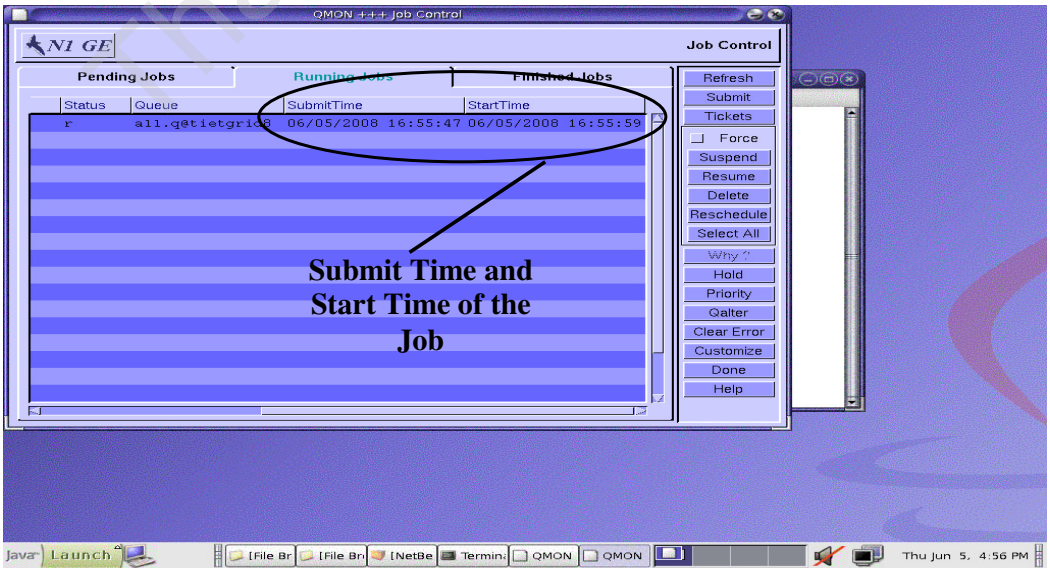
Screenshot 4.3 Submit Button on Submit Job Window

4. The job is now submitted and shown in the running jobs queue (Screenshot 4.4)



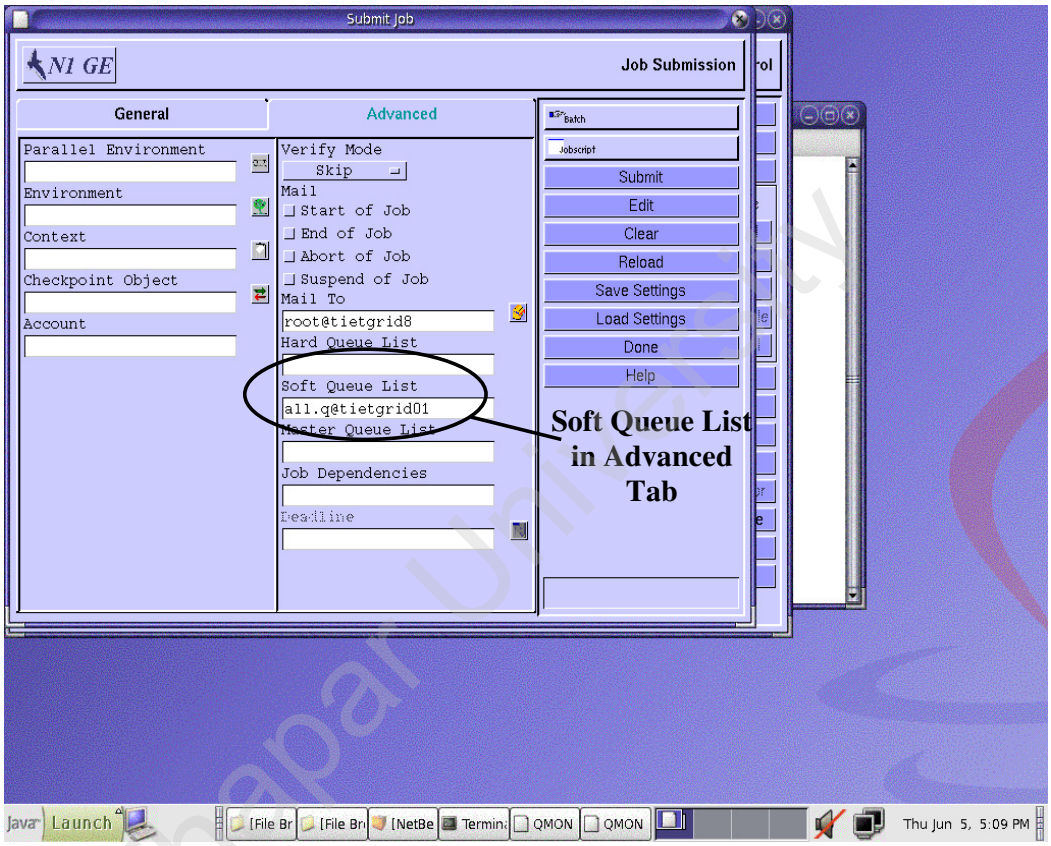
Screenshot 4.4 Job in running state in job control window

5. Scheduling time of a job is calculated as a difference between the Start Time of the job and the Submit Time of the job. The Screenshot 4.5 shows the submit time and start time of a job that is currently running. This Submit Time and Start Time of every job are noted down to calculate the Scheduling Time.



Screenshot 4.5 Submit time and start time of the currently running job

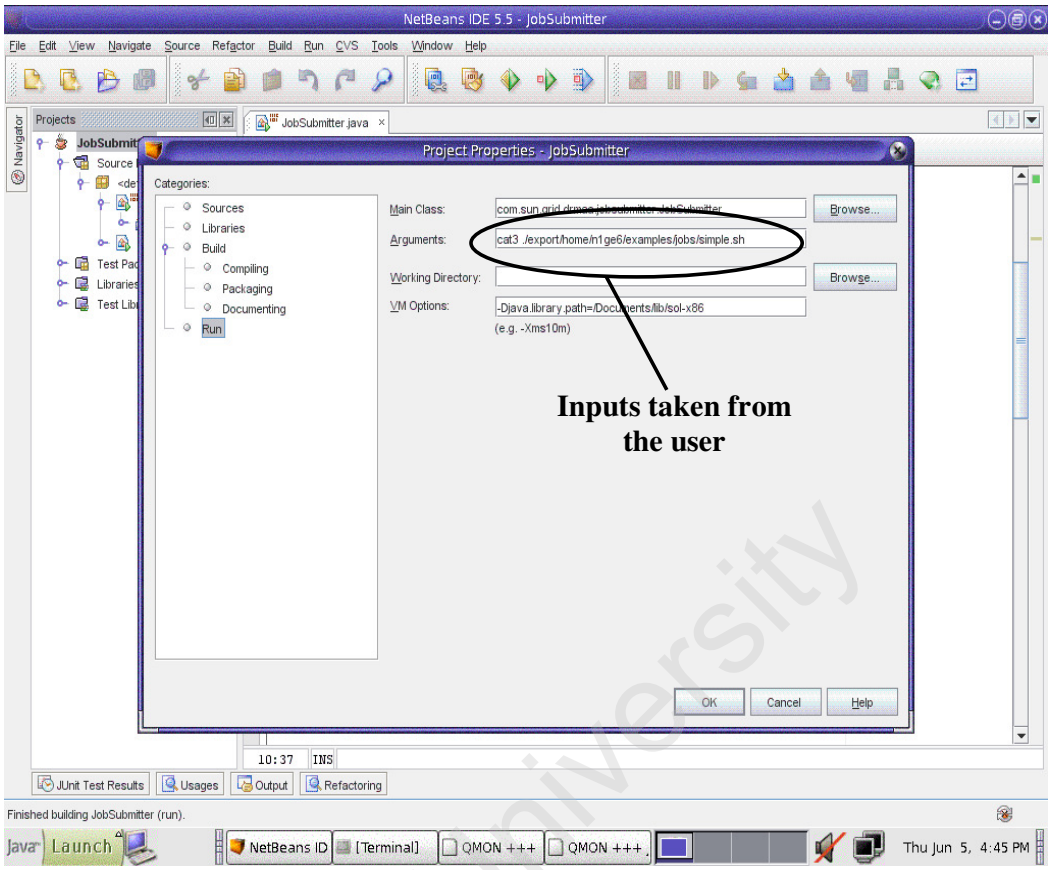
Though the QMON doesn't have the provision of submitting the jobs to a particular job resource site, but a job can be submitted to a particular queue, by specifying the particular queue in the soft queue list at the advanced tab of job submission window. Screenshots 4.6 shows this.



Screenshot 4.6 soft queue list in the advanced tab at job submission window

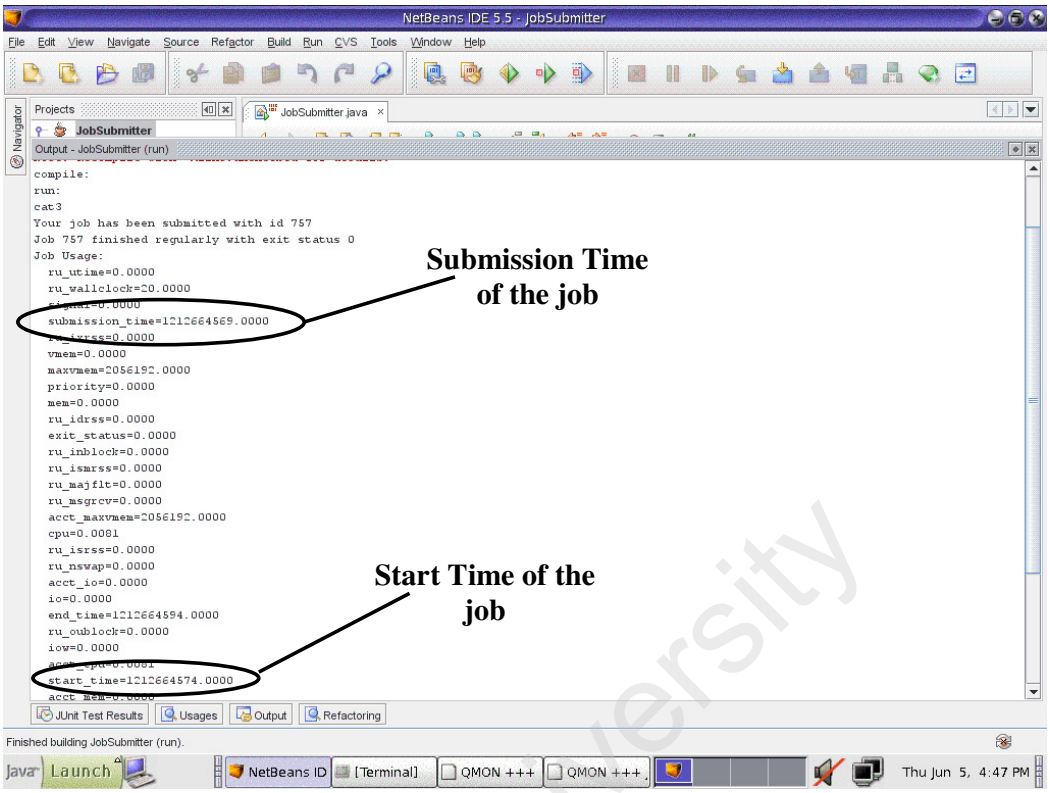
4.6 Job Submission through DRMAA Job Submitter

This section of the chapter shows that how the job is submitted through the DRMAA Job Submitter that has been developed as a solution of the problem discussed in this thesis work. The job category and the path of the job script are taken as an input from the user. The job submitter is developed using NetBeans 5.5 IDE, the screenshot 4.7 shows the inputs taken from the user.



Screenshot 4.7 Input taken from the user for DRMAA job Submitter

The DRMAA Job Submitter shows a message of successful job submission to the DRMS with job id, which has been assigned by SGE to it. After the parameters related to the job are collected and the job is finished, these parameters and the finished job status message are shown as an output of the job submitter. Screenshot 4.8 shows the parameters related to the job. The Start Time and the Submit Time of the job are noted down to calculate the Scheduling Time of the job submitted through DRMAA job Submitter. The Start Time and Submit Time parameters are shown in the output screen of the NetBeans IDE on the successful completion of the job.



Screenshot 4.8 Output screen on the successful completion of the job submitted through DRMAA Job Submitter

Chapter 5
Experimentation and Results

5.1 Assumptions and Details of Experiments

This chapter discusses the assumptions taken for the experiments that are done and the details of how the experiments are performed for comparative study of the QMON and DRMAA job Submitter.

5.1.1 Assumptions for Experiments

As the purpose of this thesis work is Performance Optimization of SGE using DRMAA on the basis of scheduling time taken by the job submitter to schedule the job therefore some assumptions are taken for the purpose of experimentation. The assumptions that are taken are listed below:

- For experimental purposes it is assumed that the queue list at the remote resource site is always available and job does not have to wait for queue to finish the pending jobs because in that case waiting time shall be added to the scheduling time of the job, which would not give the exact data that is needed for comparative study.
- The job category definitions are kept same for all remote resource sites to strengthen the comparative study based on the results that are obtained, however they can be changed without affecting the performance of the system.

5.1.2 Details of Experiments

Taking care of the assumptions that are described in the above section of this chapter, the details of experiments performed for showing the Performance Optimization of SGE using DRMAA on the basis of scheduling time of job are described here.

As discussed earlier three job categories, category1, category2 and category3 are defined for three remote resource site hosts named tietgrid01, tietgrid02 and tietgrid8 respectively and three job scripts named simple.sh, sleeper.sh and prime.sh are taken for experiments.

I. A job is submitted to each resource category site from a submit host using QMON and DRMAA Job Submitter
II. Its submit time and start time are noted down to calculate the scheduling time taken by the job.
III. The above process is repeated for each of the three job scripts being submitted to different resource category sites from a submission host.
IV. This is done several times to get an average scheduling time of each job for a particular resource category site submitted by a submission host.
V. The average scheduling time obtained for each job script being executed on different job category sites submitted through QMON and DRMAA Job Submitter are compared to show the Performance Optimization that has been achieved by DRMAA Job Submitter over QMON.

The above steps can be better explained by taking a real life example. The job script prime.sh is submitted from a submission host tietgrid8 to remote resource site category1 through QMON by specifying it in soft queue list and through DRMAA Job Submitter by just asking the user the job category. Then the scheduling time for both the job submitters are calculated for comparison. The above procedure is repeated several times and average scheduling time in each of the case for QMON and DRMAA Job Submitter is considered for obtaining the final results and comparisons.

5.2 Analysis of Job Submission though QMON

Each job script prime.sh, simple.sh and sleeper.sh is submitted to predefined resource category sites categrory1, category2 and category3 from the submission host tietgrid8 through QMON and the average Scheduling Time is calculated by taking difference of Submit Time and Start Time. The results obtained are shown in following tables.

These results are to be compared with the results obtained from DRMAA Job Submitter.

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	12:06:45	12:06:55	10
2.	12:08:05	12:08:10	5
3.	12:16:01	12:16:10	9
4.	12:17:13	12:17:25	12
5.	12:27:53	12:27:55	2
6.	12:35:27	12:35:40	13
7.	12:44:35	12:44:40	5
8.	12:53:36	12:53:40	4
9.	12:54:16	12:54:25	9
10.	13:02:21	13:02:25	4
11.	13:03:17	13:03:25	8
12.	13:12:02	13:12:10	8
13.	13:12:34	13:12:40	6
Average Job Scheduling Time			**7.3**

Table 5.1 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category1" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	12:21:45	12:21:53	8
2.	12:32:13	12:32:23	10
3.	12:34:04	12:34:08	4
4.	12:46:20	12:46:23	3
5.	12:47:40	12:47:53	13
6.	13:14:17	13:14:23	6
7.	13:23:56	13:24:08	12
8.	13:29:17	13:29:23	6
9.	13:33:51	13:33:53	2
10.	13:38:12	13:38:23	11
11.	13:44:14	13:44:23	9
12.	13:47:38	13:47:53	5
13.	13:53:33	13:53:38	5
14.	13:57:07	13:57:08	1
Average Job Scheduling Time			**6.7**

Table 5.2 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category2" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	12:16:24	12:16:38	14
2.	16:15:42	16:15:53	12
3.	16:37:56	16:38:08	12
4.	16:48:23	16:48:38	15
5.	16:15:12	16:15:23	11
6.	17:34:30	17:34:38	8
7.	17:49:54	17:50:08	14
8.	11:10:45	11:10:56	11
9.	11:21:48	11:21:56	8
10.	12:05:08	12:05:20	12
Average Job Scheduling Time			**10.6**

Table 5.3 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category3" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:33:35	11:33:40	5
2.	11:34:15	11:34:25	10
3.	11:34:50	11:34:55	5
4.	11:35:18	11:35:25	7
5.	11:36:11	11:36:25	14
6.	11:36:57	11:37:10	13
7.	11:37:35	11:37:40	5
8.	11:38:42	11:38:55	13
9.	11:40:44	11:40:55	10
10.	11:41:46	11:41:55	9
11.	11:42:18	11:42:25	7
Average Job Scheduling Time			**8.8**

Table 5.4 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category1" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:34:40	11:34:53	13
2.	11:35:25	11:35:38	13
3.	11:36:46	11:36:53	7
4.	11:37:26	11:37:38	12
5.	11:45:14	11:45:23	9
6.	11:50:46	11:50:53	7
7.	11:52:11	11:52:23	12
8.	11:53:42	11:53:53	11
9.	11:56:13	11:56:23	10
10.	11:58:44	11:58:53	9
11.	11:59:29	11:59:38	9
12.	12:00:18	12:00:23	5
Average Job Scheduling Time			**9.5**

Table 5.5 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category2" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	13:12:54	13:13:01	7
2.	13:13:56	13:14:01	5
3.	13:15:34	13:15:46	12
4.	13:17:35	13:17:46	11
5.	13:19:06	13:19:16	10
6.	13:19:08	13:19:16	8
7.	13:22:51	13:23:01	10
8.	13:23:53	13:24:01	8
9.	13:24:34	13:24:46	12
10.	13:26:08	13:26:16	10
11.	13:27:10	13:27:16	6
12.	13:27:52	13:28:01	9
13.	13:28:27	13:28:31	4
Average Job Scheduling Time			**8.6**

Table 5.6 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category3" QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	17:30:11	17:30:20	9
2.	17:31:06	17:31:20	14
3.	17:32:23	17:32:35	12
4.	17:33:54	17:34:05	11
5.	17:34:43	17:34:50	7
6.	17:36:14	17:36:20	6
7.	17:37:09	17:39:20	11
8.	17:37:54	17:38:05	11
9.	17:39:38	17:39:50	12
10.	17:40:59	17:41:05	6
11.	17:41:06	17:41:20	14
12.	17:42:10	17:42:20	10
13.	17:42:45	17:42:50	5
14.	17:43:56	17:44:05	9
Average Job Scheduling Time			**9.7**

Table 5.7 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category1" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	10:55:54	10:56:08	14
2.	10:57:19	10:57:23	4
3.	10:58:29	10:58:38	9
4.	10:59:15	10:59:23	8
5.	11:00:28	11:00:38	10
6.	11:01:09	11:01:23	14
7.	11:02:05	11:02:08	3
8.	11:02:33	11:02:38	5
9.	11:03:53	11:04:08	15
10.	11:04:41	11:04:53	12
11.	11:05:26	11:05:38	12
12.	11:07:02	11:07:08	6
Average Job Scheduling Time			**8.8**

Table 5.8 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category2" through QMON

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:53:34	11:53:47	13
2.	12:00:06	12:00:17	11
3.	12:01:39	12:01:47	8
4.	12:04:24	12:04:32	8
5.	12:05:47	12:06:02	15
6.	12:07:20	12:07:32	12
7.	12:09:32	12:09:47	15
8.	12:15:06	12:15:17	11
9.	12:16:32	12:16:47	15
10.	12:17:53	12:18:02	9
11.	12:20:21	12:20:32	11
12.	12:23:51	12:24:02	11
Average Job Scheduling Time			**10.3**

Table 5.9 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category3" through QMON

The results shown in Table 5.10 are based on the data shown in Tables 5.1 - 5.9.

Average Scheduling Time In Seconds (QMON)			
Job Category / Job Script	*Category1*	*Category2*	*Category3*
prime.sh	7.3	6.7	10.6
simple.sh	8.8	9.5	8.4
sleeper.sh	9.7	8.8	10.3

Table 5.10 Average Job Scheduling Time of jobs submitted through QMON

5.3 Analysis of Job Submission through DRMAA Job Submitter

The each job scripts prime.sh, simple.sh and sleeper.sh are submitted to predefined resource category sites categrory1, category2 and category3 from the submission host tietgrid8 through DRMAA Job Submitter and the average Scheduling Time is calculated by taking difference of Submit Time and Start Time. The results obtained are shown in the following tables.

Average Scheduling Time In Seconds (DRMAA Job Submitter)			
Job Category / Job Script	*Category1*	*Category2*	*Category3*
prime.sh	5.7	5.2	9.1
simple.sh	4.3	4.1	3.3
sleeper.sh	6.1	4.6	6.5

Table 5.11 Average Scheduling Time of jobs submitted through DRMAA Job Submitter

The results shown in Table 5.11 are based on results shown in Tables 5.12 – 5.20

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	15:11:22	15:11:29	7
2.	15:12:09	15:12:14	5
3.	15:20:44	15:20:59	5
4.	15:21:11	15:21:14	3
5.	15:30:07	15:30:14	7
6.	15:30:23	15:30:29	6
7.	15:40:35	15:40:44	9
8.	15:40:58	15:40:59	1
9.	10:47:37	10:47:41	4
10.	10:56:33	10:56:41	8
11.	10:56:37	10:56:41	4
12.	11:10:25	11:10:26	1
13.	11:10:31	11:10:41	10
14.	11:24:46	11:24:56	10
Average Job Scheduling Time			**4.6**

Table 5.12 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category1" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	14:02:47	14:02:53	6
2.	14:06:04	14:06:08	4
3.	14:12:35	14:12:38	3
4.	14:17:15	14:17:23	8
5.	14:23:00	14:23:08	8
6.	14:27:00	14:27:08	8
7.	14:31:52	14:31:53	1
8.	15:02:34	15:02:38	4
9.	15:09:36	15:09:38	2
10.	15:11:31	15:11:38	7
11.	15:19:16	15:19:23	7
Average Job Scheduling Time			**5.2**

Table 5.13 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category2" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	12:28:07	12:28:20	13
2.	12:49:12	12:49:20	8
3.	12:49:57	12:50:05	8
4.	13:19:47	13:19:50	3
5.	15:46:25	15:46:35	10
6.	15:50:51	15:51:05	14
7.	15:58:07	15:58:20	13
8.	16:04:16	16:04:20	4
9.	16:07:31	16:07:35	4
10.	16:15:52	16:16:05	13
11.	16:19:37	16:19:50	13
12.	16:26:43	16:26:50	7
13.	16:31:18	16:31:20	2
Average Job Scheduling Time			**9.1**

Table 5.14 Results obtained for job script "prime.sh" submitted to Resource Job Category Site "Category3" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:45:06	11:45:10	4
2.	11:45:38	11:45:40	2
3.	11:46:17	11:46:25	8
4.	11:47:38	11:47:40	2
5.	11:48:19	11:48:25	6
6.	11:48:49	11:48:55	6
7.	11:49:19	11:49:25	6
8.	11:50:05	11:50:10	5
9.	11:52:06	11:52:10	4
10.	11:52:34	11:52:40	6
11.	11:53:07	11:53:10	3
12.	11:54:06	11:54:10	4
13.	11:54:39	11:54:40	1
Average Job Scheduling Time			**4.3**

Table 5.15 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category1" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	12:10:51	12:10:53	2
2.	12:11:16	12:11:23	5
3.	12:12:51	12:12:53	2
4.	12:14:14	12:14:23	9
5.	12:14:48	12:14:53	5
6.	12:15:21	12:15:23	2
7.	12:15:47	12:15:53	6
8.	12:16:21	12:16:23	2
9.	12:16:49	12:16:53	4
10.	12:17:22	12:17:23	1
11.	12:17:51	12:17:53	2
12.	12:18:18	12:18:23	5
13.	12:18:44	12:18:53	9
Average Job Scheduling Time			**4.1**

Table 5.16 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category2" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	13:32:27	13:32:31	4
2.	13:33:00	13:33:01	1
3.	13:33:39	13:33:46	7
4.	13:34:59	13:35:01	2
5.	13:35:28	13:35:31	3
6.	13:35:58	13:36:01	3
7.	13:36:29	13:36:31	2
8.	13:37:42	13:37:46	4
9.	13:38:13	13:38:16	3
10.	13:38:44	13:38:46	2
11.	13:40:25	13:40:31	6
Average Job Scheduling Time			**3.3**

Table 5.17 Results obtained for job script "simple.sh" submitted to Resource Job Category Site "Category3" through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	17:47:14	17:47:20	6
2.	17:47:54	17:48:05	11
3.	17:50:02	17:50:05	3
4.	17:50:47	17:50:50	3
5.	17:51:19	17:51:20	1
6.	17:53:02	17:53:05	12
7.	17:56:55	17:56:05	3
8.	17:56:55	17:57:05	10
9.	17:58:32	17:58:35	3
10.	17:59:09	17:59:20	11
11.	18:00:46	18:0050	4
12.	18:02:40	18:00:50	10
Average Job Scheduling Time			**6.1**

Table 5.18 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category1" submitted through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:09:34	11:09:38	4
2.	11:10:21	11:10:23	2
3.	11:10:52	11:10:53	1
4.	11:22:15	11:22:23	8
5.	11:23:01	11:23:08	7
6.	11:23:49	11:23:53	4
7.	11:24:18	11:24:23	5
8.	11:25:35	11:25:38	3
9.	11:30:01	11:30:08	7
10.	11:30:36	11:30:38	2
11.	11:31:15	11:31:23	8
Average Job Scheduling Time			**4.6**

Table 5.19 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category2" submitted through DRMAA Job Submitter

Sr. no.	Submission Time	Start Time	Scheduling Time in seconds
1.	11:30:57	11:31:02	5
2.	11:35:42	11:35:47	5
3.	11:38:42	11:38:47	5
4.	11:39:57	11:40:02	5
5.	11:40:53	11:41:02	9
6.	11:42:10	11:42:17	7
7.	11:44:39	11:44:47	8
8.	11:45:56	11:46:02	6
9.	11:48:40	11:48:47	7
10.	11:50:09	11:50:17	8
Average Job Scheduling Time			**6.5**

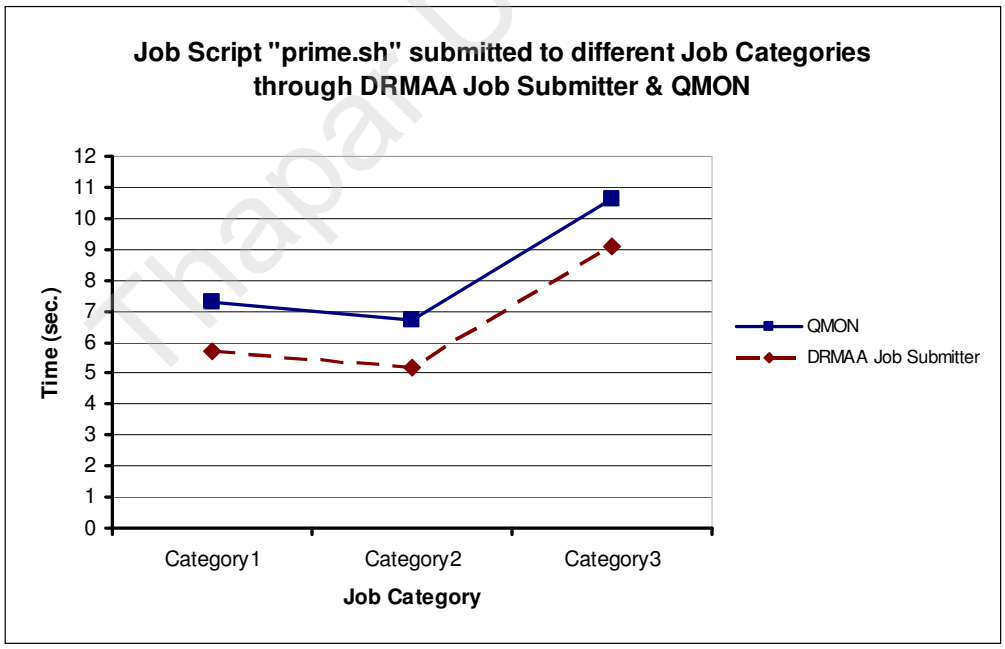
Table 5.20 Results obtained for job script "sleeper.sh" submitted to Resource Job Category Site "Category3" submitted through DRMAA Job Submitter

5.4 Results

This section shows the results obtained from the experiments that are performed. The results and the graphs clearly show the improvement in scheduling time of the job compared to QMON and DRMAA Job Submitter.

Performance Optimization on "prime.sh" Job Script			
Job Category	**Scheduling Time in sec. (QMON)**	**Scheduling Time in sec. (DRMAA Job Submitter)**	**Difference (Improvement in Scheduling Time)**
Category1	7.3	5.7	1.6
Category2	6.7	5.2	1.5
Category3	10.6	9.1	1.5

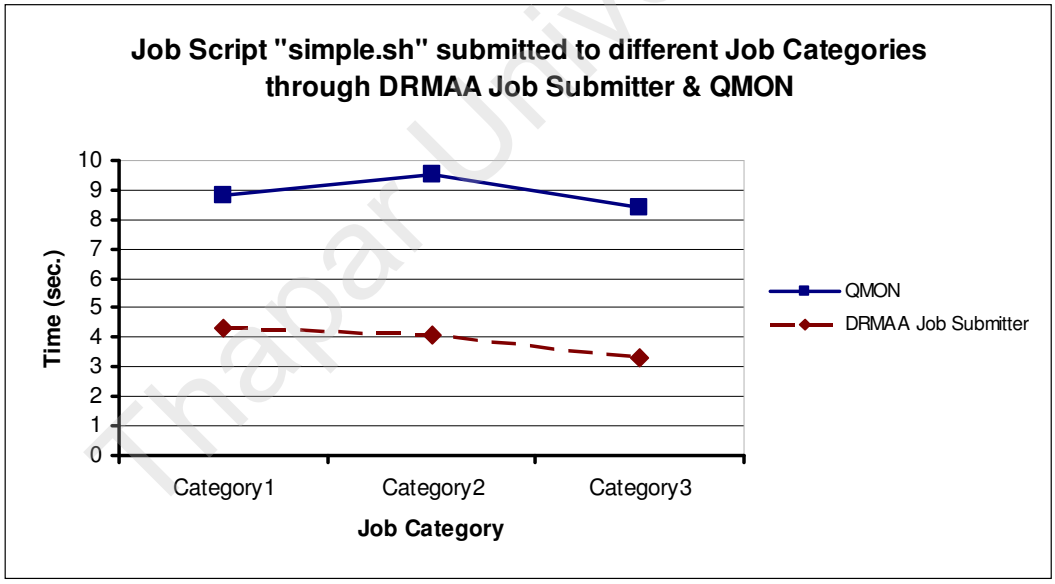
Table 5.21 Scheduling Time for "prime.sh" on different Job Category Sites



Graph 5.1 Improvements in Scheduling Time for "prime.sh"

Performance Optimization on "simple.sh" Job Script			
Job Category	**Scheduling Time in sec. (QMON)**	**Scheduling Time in sec. (DRMAA Job Submitter)**	**Difference (Improvement in Scheduling Time)**
Category1	8.8	4.3	4.5
Category2	9.5	4.1	5.4
Category3	8.4	3.3	5.1

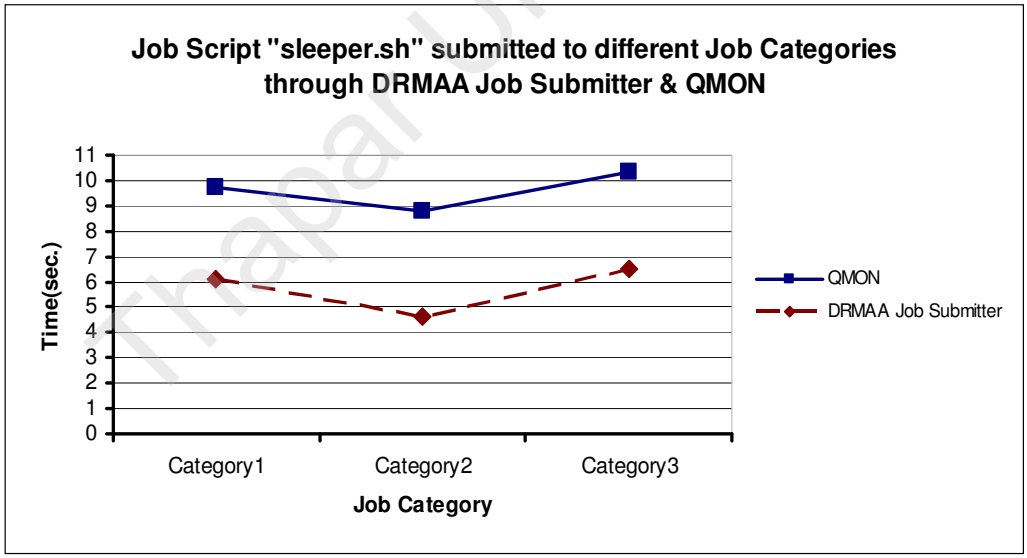
Table 5.22 Scheduling Time for "simple.sh" on different Job Category Sites



Graph 5.2 Improvements in Scheduling Time for "simple.sh"

Performance Optimization on "sleeper.sh" Job Script			
Job Category	**Scheduling Time in sec. (QMON)**	**Scheduling Time in sec. (DRMAA Job Submitter)**	**Difference (Improvement in Scheduling Time)**
Category1	9.7	6.1	3.6
Category2	8.8	4.6	4.2
Category3	10.3	6.5	3.8

Table 5.23 Scheduling Time for "sleeper.sh" on different Job Category Sites



Graph 5.3 Improvements in Scheduling Time for "sleeper.sh"

Chapter 6
Conclusion and Future Work

This thesis work has given an insight of grid computing and the role played by the middleware in the working of a grid. It is evident from this thesis work that job submission portals or the interfaces if developed and used efficiently can be used to improve the performance of the middleware.

6.1 Conclusion

In this thesis multiple aspects of a job submitter interface are discussed and the focus has been on improving the performance of SGE using DRMAA. The successful development and implementation of a DRMAA Job Submitter has helped to overcome the problems and limitations of QMON such as, not being able to know the resource requirement of job at the time of submission and submit the job to a predefined particular resource job category.

As a proposed solution of the problems observed a new job submitter has been developed, which improves the performance of the middleware SGE.

The new job submitter for SGE has been developed using Distributed Resource Management Application API (DRMAA) with Java Bindings. It has been successfully implemented and experiments performed on it show the improvements in job scheduling time.

The scheduling time of the job submitted through QMON and DRMAA Job Submitter is compared and graphs generated show the improvements in the performance based on the job scheduling time.

6.2 Future Scope

Improvement in the performance of the system is always desirable and it has been achieved in this thesis work. The performance of the middleware can be further

improved and the Job Submitter can be further enhanced by:

- Including job submission for parallel jobs.
- Adding bulk job submission option to it.
- Rescheduling the job to other job category sites if the resource category site is not available.
- Including some mechanism that a job does not have to die, starving to get resources.

References

[1] Grid Computing, Joshy Joseph, Craig Fellenstein, Publisher: Prentice Hall PTR Pub Date: December 30, 2003, ISBN: 0-13-145660-1

[2] Introduction, What is a grid? http://www.sura.org/cookbook/gtcb/index.php

[3] Jean-Christophe Durand, "Grid Computing: A Conceptual and Practical Study", Master's Thesis submitted at University of Lausanne, November 8, 2004

[4] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke, "The physiology of the Grid" Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[5] Chaitanya Kandagatla, "Survey and Taxonomy of Grid Resource Management Systems", University of Texas, Austin, 2003
http://www.cs.utexas.edu/users/browne/cs395f2003/projects/KandagatlaReport.pdf

[6] IBM Grid solutions: http://www-1.ibm.com/grid/solutions/index.shtml

[7] C. Kesselman. I. Foster. Computational grids. In The Grid: Blueprint for a New Computing Infrastructure., chapter 2. Morgan-kaufman edition, 1999.

[8] National collaboratories: Applying information technology for scientific research. In *National Academy Press*. 1993.

[9] Ann Chervenak and others, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets", Journal of Network and Computer Applications, 2001

[10] Rajkumar Buyya and Srikumar venugopal , "A Gentle Introduction to Grid Computing and Technologies", http://www.buyya.com/papers/GridIntro-CSI2005.pdf

[11] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Alchemi: A .NET-Based Enterprise Grid Computing System", Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005, Las Vegas, USA.

[12] Gregor von laszewaski, Ian Foster, Argonne National Laboratory, 9700 South Cass Avenue, Argonne IL 60439, U.S.A, Designing Grid Based Problem solving Environments www-unix.mcs.anl.gov/~laszewsk/papers/cog-pse-final.pdf

[13] Francois Grey, Matti Heikkurinen, Rosy Mondardini, Robindra Prabhu, "Brief History of Grid" http://Gridcafe.web.cern.ch/Gridcafe/Gridhistory/history.html

[14] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova, GridRPC: A Remote Procedure Call API for Grid Computing, July 2002

[15] https://forge.gridforum.org/projects/gridrpc-wg/ – GridRPC's work group homepage

[16] SAGA (http://www.ogf.org/gf/group_info/view.php?group=saga-rg)

[17] Urszula Herman-Izycka, Michal Ejdys, Kevin Huguenin "APIs and End Users Cluster and Grid Computing 2006" June 9, 2006

[18] DRMAA (http://www.drmaa.org)

[19] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Alchemi: NET-based Grid Computing Framework and its Integration into Global Grids" Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2003. http://www.alchemi.net/files/alchemi_techreport.pdf

[20] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal, "Peerto- Peer Grid Computing and a .NET-based Alchemi - Framework". Grid Computing and Distributed Systems (GRIDS) Laboratory Department of

Computer Science and Software Engineering, The University of Melbourne, Australia. http://www.gridbus.org/papers/Alchemi.pdf

[21] Krishna Nadiminti, Akshay Luther, Rajkumar Buyya, Alchemi: A .NET-based Enterprise Grid System and Framework, User Guide for Alchemi 1.0 December 2005.

[22] R. Buyya, Economic-based Distributed Resource Management and Scheduling for Grid Computing, PhD Thesis, Monash University, Melbourne, Australia, April 2002.

[23] M. Litzkow, M. Livny, M. Mutka, "Condor---a hunter of idle workstations, in: Proceedings of the Eighth International Conference of Distributed Computing Systems", June 1988.

[24] Amarjit Manjotra, Ms. Indeveer Channa, "Quantifying Impacts of Resource Heterogeneity on Grid Performance" thesis submitted at Computer Science and Engineering Department, Thapar Institute of Engineering and Technology (Deemed University), Patiala, May 2006.

[25] Prof. Richard Sinnott Dr. John Watt, Tutorial 1: Introduction to Globus Toolkit.

[26] N1 Grid Engine 6 User's Guide, Sun Microsystems, Inc. 4150 Network Circle Santa Clara, CA 95054 U.S.A. PartNo: 820–0699, May 2007

[27] Andreas Haas, Senior Staff Engineer, N1™ Grid Engine Engineering, Sun Microsystems, Co-Chair of DRMAA Working, Group Global Grid Forum, "SHORT DRMAA PRIMER".
http://www.sun.com/software/gridware/drmaa_primer.pdf

[28] Daniel Templeton, Staff Engineer, Sun Grid Engine, Sun Microsystems Inc, "Advanced Sun Grid Administration Guide".
https://www.middleware.georgetown.edu/confluence/display/HPCT/Advanced+Sun+Grid+Engine+Configuration+and+Administration

[29] Andreas Haas, Senior Staff Engineer, Sun Microsystems GmbH, "Sun Grid Engine 6.1 + DRMAA interface".
http://gks07.fzk.de/Talks/Friday/2007_09_14__10_00_Haas_Sun.pdf

[30] Roger Brobst, Cadence Design Systems Waiman Chan, IBM Fritz Ferstl, Sun Microsystems Jeff Gardiner, John P. Robarts Res. Inst., GFD-R.022 Distributed Resource Management Application API (DRMAA) Working Group, April, 2004 Updated Aug, 2007.

Appendix – A

Installation of Sun N1 Grid Engine

1. Install Solaris 10

2. Create the installation directory as sge-root in which contents of the distribution media will be loaded.

3. Install the binaries for all binary architectures that are to be used by any of your master, execution, and submit hosts in your grid engine system cluster.

   The pkgadd Method

   The pkgadd format is provided for the Solaris Operating System. To facilitate remote installation, the pkgadd directories are also provided in zip files.

   Install the following packages:

   _ SUNWsgeec – Architecture independent files

   _ SUNWsgeed – Documentation

   _ SUNWsgee – Solaris (SPARC platform) 32-bit binaries for Solaris 7, Solaris 8, and Solaris 9 Operating Systems

   _ SUNWsgeex – Solaris (SPARC platform) 64-bit binaries for Solaris 7, Solaris 8, and Solaris 9 Operating Systems

   _ SUNWsgeei — Solaris (x86 platform) binaries for Solaris 8 and Solaris 9 Operating Systems

   _ SUNWsgeeax - Solaris (x64 platform) binaries for Solaris 10 Operating System

   _ SUNWsgeea - Accounting and Reporting Console (ARCo) packages for the Solaris and Linux Operating systems.

At the command prompt, type the following commands, responding to the script questions.

cd cdrom_mount_point/N1_Grid_Engine_6u4

pkgadd -d ./Common/Packages/SUNWsgeec

pkgadd -d ./Docs/Packages/SUNWsgeed

pkgadd -d ./Solaris_sparc/Packages/SUNWsgee

pkgadd -d ./Solaris_sparc/Packages/SUNWsgeex

pkgadd -d ./Solaris_x86/Packages/SUNWsgeei

pkgadd -d ./Solaris_x64/Packages/SUNWsgeeax

4. Install Grid Engine software interactively.

Install Master host

1. Log in to the master host as root.

2. Ensure that the $SGE_ROOT environment variable is set by typing:

echo $SGE_ROOT

- If the $SGE_ROOT environment variable is not set, set it now, by typing:

SGE_ROOT=sge-root; export SGE_ROOT

3. Change to the installation directory.

- If the directory where the installation files reside is visible from the master host, change directories (cd) to the installation directory sge-root, and then proceed to Step 4.
- If the directory is not visible and cannot be made visible, do the following:

a. Create a local installation directory, sge-root, on the master host.

b. Copy the installation files to the local installation directory sge-root across the network (for example, by using ftp or rcp).

c. Change directories (cd) to the local sge-root directory.

4. Type the install_qmaster command.

This command starts the master host installation procedure. We are asked several questions, and may need to run some administrative actions.

% ./install_qmaster

5. Choose an administrative account owner.

sgeadmin.

6. Verify the sge-root directory setting.

7. Enter the name of your cell.

CoEPDC

8. Specify a spool directory.

/opt/n1ge6/default/spool/qmaster is the qmaster spool directory by default

9. Specify whether you want to use classic spooling or Berkeley DB.

10. Enter a group ID range 20000-20100

11. Verify the spooling directory for the execution daemon.

Default: [/opt/n1ge6/default/spool]

12. Enter the email address of the user who should receive problem reports.

msingh@thaper.edu

13. Identify the hosts that you will later install as execution hosts.

Install Execution host

1. Log in to the execution host as root.

2. As you did for the master installation, either copy the installation files to a local installation directory sge-root or use a network installation directory.

   Also copy the cell name (CoEPDC) folder, lib and util folder from the system where master host was installed.

3. Ensure that you have set the $SGE_ROOT environment variable by typing:

   # echo $SGE_ROOT

   If the $SGE_ROOT environment variable is not set, set it now, by typing:

   # SGE_ROOT=sge-root; export SGE_ROOT

4. Change directory (cd) to the installation directory, sge-root.

5. Verify that the execution host has been declared on the administration host.

   # qconf -sh

6. Run the install_execd command.

   % ./install_execd

   This command starts the execution host installation procedure.

7. Verify the sge-root directory setting.

8. Enter the name of your cell. CoEPDC

9. The install script checks to see if the admin user already exists. If the admin user already exists, the script continues uninterrupted otherwise the script shows a screen where you must supply a password for the admin user. After the admin user is created, press Enter to continue with the installation.

10. Specify whether you want to use a local spool directory.

11. Specify whether you want execd to start automatically at boot time.

12. Specify a queue for this host.

Appendix – B

Using DRMAA with NetBeans 5.x

To use the DRMAA classes with your NetBeans 5.0 or 5.5 project, follow these steps:

1. Click mouse button 3 on the project node and select Properties.
2. Determine whether your project generates a build file or uses an existing file.
    - If your project uses a generated build file:

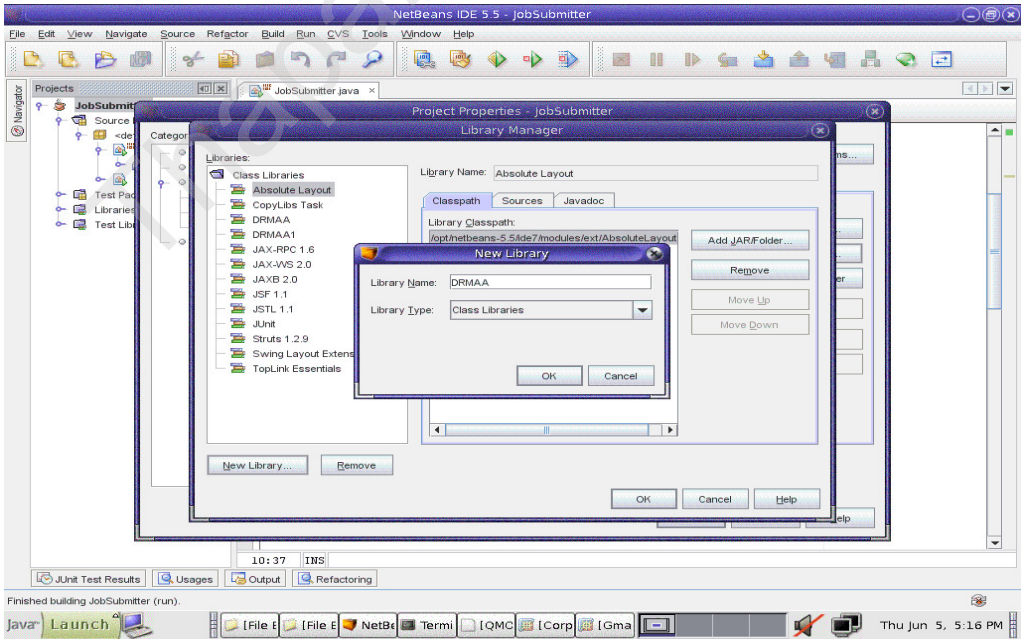
      a. Select Libraries in the left column.

      b. Click Add Library.

      c. Click Manage Libraries in the Libraries dialog box.

      d. Click New Library in the Library Management dialog box.

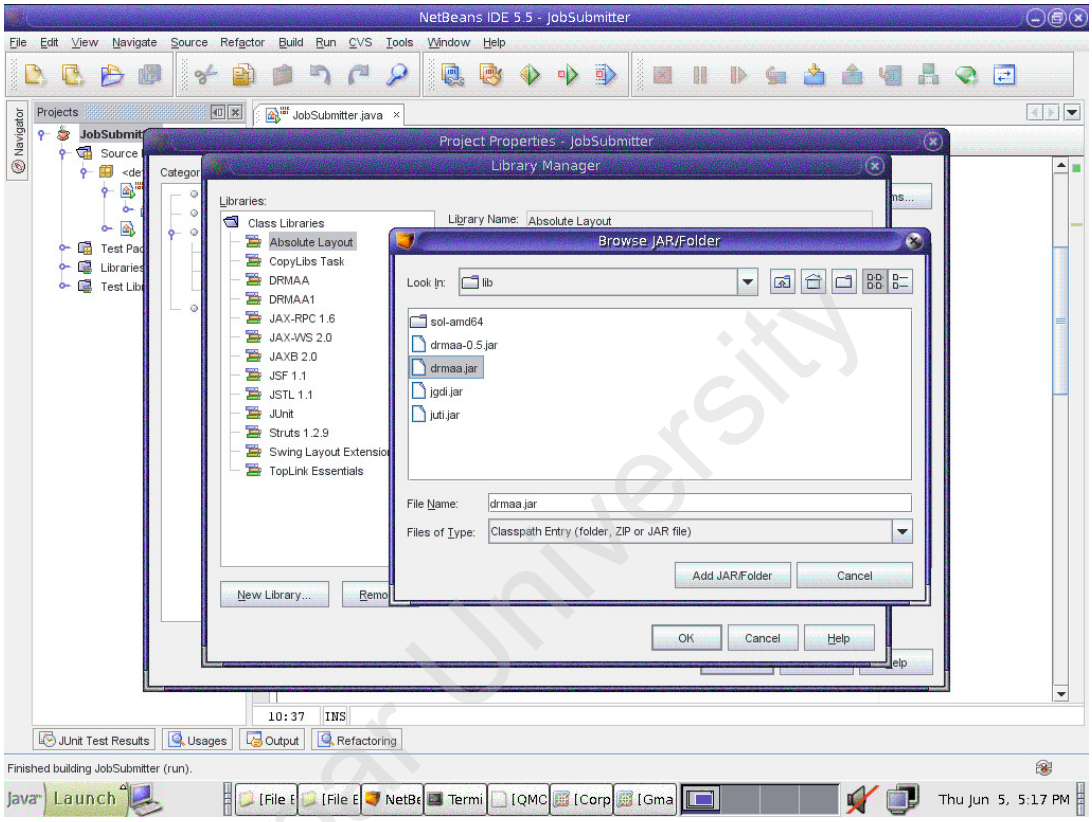
      e. Type **DRMAA** in the Library Name field in the New Library dialog box.



f. Click OK to dismiss the New Library dialog box.

g. Click Add JAR/Folder.

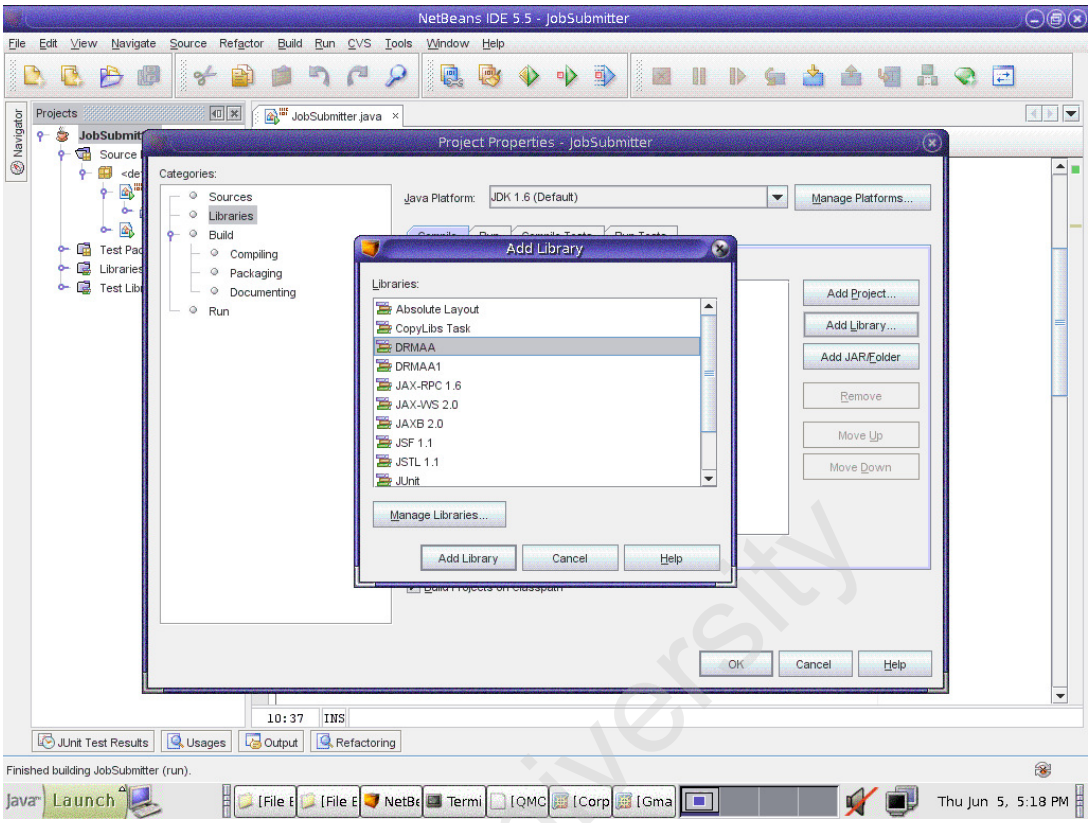
h. Browse to the *sge-root*/lib directory in the file chooser dialog box and select the drmaa.jar file.



i. Click Add JAR/Folder to dismiss the file chooser dialog box.

j. Click OK to dismiss the Library Management dialog box.

k. Select the DRMAA library and click Add Library to dismiss the Libraries dialog box.



- If your project uses an existing build file:

  a. Select Java Sources Classpath in the left column.

  b. Click Add JAR/Folder.

  c. Browse to the *sge-root*/lib directory in the file chooser dialog box and select the drmaa.jar file.

  d. Click Choose to dismiss the file chooser dialog box.

3. Click OK to dismiss the properties dialog box.
4. Verify that the DRMAA shared library is in the library search path.

   To run your application from NetBeans, the DRMAA shared library file *sge-root*/lib/*arch*/libdrmaa.so must be included in the library search path (LD_LIBRARY_PATH on the Solaris Operating Environment and Linux). The *sge-root*/lib/*arch* directory is *not* included automatically when you set your

environment using the settings.sh or settings.csh files.To set up the path for the shared library, perform one of the following:

- . Set up your environment in the shell before launching NetBeans.
- Add to the*netbeans-root*/etc/netbeans.conf file to set up the environment, such as:

Setup environment for SGE

. <sge-root>/<sge_cell>/common/settings.sh

ARCH=‘$SGE_ROOT/util/arch‘

LD_LIBRARY_PATH=$SGE_ROOT/lib/$ARCH; export LD_LIBRARY_PATH

Running Your Java Application

To run your compiled DRMAA application, verify the following:

- . The *sge-root*/lib/*arch* directory must be included in the library search path (LD_LIBRARY_PATH on the Solaris Operating Environment and Linux). The *sge-root*/lib/*arch* directory is *not* included automatically when you set your environment using the settings.sh or settings.csh files.
- You must be logged into a machine that is an N1 Grid Engine submit host. If the machine is not an N1 Grid Engine submit host, all DRMAA method calls will fail, throwing a

DrmCommunicationException.

Papers Accepted / Communicated

1. Shilpi Gupta, Seema Bawa, "Optimizing Performance of a Grid Middleware using DRMAA" at **2008 IEEE Asia-Pacific Services Computing Conference (IEEE APSCC 2008)** to be held in Jiaosi, Yilan, Taiwan from December 9-12, 2008 **[Communicated].**