

MULTIAGENT FRAMEWORK FOR INTERACTIVE JOB MANAGEMENT FOR GRID

BY

SNEHA MEHTA

07MCE009



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AHMEDABAD-382481

May 2009

MULTIAGENT FRAMEWORK FOR INTERACTIVE JOB MANAGEMENT FOR GRID

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

by

SNEHA MEHTA

07MCE009



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AHMEDABAD-382481**

May 2009

Certificate

This is to certify that the Major Project entitled "MultiAgent Framework for Interactive Job Management for Grid" submitted by Mehta Sneha L (07MCE009), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Madhuri Bhavsar
Guide, Associate Professor,
Department Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

Modern science and technology are collaborating with each other to achieve the goals, performance and greater impact all over the globe. These collaborations are multi-institutional, multi-disciplinary and geographically distributed environments. A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high end computational capabilities.

Intelligent MultiAgent approaches are well-suited for many types of services. Intelligent cooperative agents enable the system to autonomously adapt to users computation needs as well as dynamically changing computing resource environments.

The application domains addressed by Grid technologies need to be extended to include graphical, interactive sessions. Propose interactive grids - next generation grids addressing the needs of graphical, interactive sessions. Interactive Grids permit end-users to access and control a remote resource eg. remote workstation in the Grid for graphical, interactive use. Such an interactive session on a remote workstation can be used for many applications.

The motive of this project is to elaborate the effectiveness of Grid computing by Interactive Agent based Job management. This special issue brings together fields of grid computing and multiagent technology together with providing interactivity to present their work on the applications of agent-based techniques and approaches in managing and allocating resources in grid computing environment.

The various phases of the project carried out are configuration of Grid, Generation of Super Computing power by connecting the number of nodes to the grid, Configuration of heterogeneous grid environment Agent implementation and finally agent based interactive job management through grid services.

Acknowledgements

With immense pleasure, I would like to present the project report on "MultiAgent Framework for Interactive Job Management for Grid". I am very much grateful to all the people who have helped me and provided guidance during the course of the project.

First of all, I would like to thank Prof. Madhuri Bhavsar, Institute of Technology, Nirma University, and Ahmedabad, who persuaded in taking this particular project and constantly supported and motivated to complete the project. Her keen understanding and knowledge of the grid concepts helped in understanding the grid architecture and workflow, which made it easy for me to implement the same.

I would certainly like to thank Dr. S.N.Pradhan, PG-Coordinator, Computer Engineering Department, Institute of Technology, and Ahmedabad for his constant encouragement and motivation throughout the course of the project.

Finally, I am thankful to Nirma University for providing all the required resources that were needed for the project. I would also like to thank all those people who have directly or indirectly helped us in our project.

- Mehta Sneha L
07MCE009

List of Figures

2.1	View of MultiAgent Framework	7
3.1	Globus Architecture	11
3.2	GRAM Architecture	13
5.1	Overview Of Interactive Grid Computing System	22
6.1	Agent Based Grid System hierarchy	26
6.2	MultiAgent Design	27
7.1	job processing sequence	30
7.2	Job execution using grid services	32
7.3	Job Execution Sequence	33
8.1	Condor Architecture Combined with Grid	35
8.2	Generation of Super Computing Power	43
9.1	Job Management System Architecture	46
9.2	Representation of Agent Based System	47
9.3	UserLogin	48
9.4	welcome	49
9.5	Performance Comparison of System having Agents and without Agents	50
9.6	Create proxy	51
9.7	Job Submission	52
9.8	Job Checkpointing	53
9.9	Job Termination	53

Chapter 1

Introduction

1.1 Orientation of Project

Grid computing is gaining a lot of attention within the IT industry. Though it has been used within the academic and scientific community for some time, standards, enabling technologies, toolkits and products are becoming available that allow businesses to utilize and reap the advantages of grid computing [1].

Grid Computing is an active research area which promises to provide a flexible infrastructure for complex, dynamic and distributed resource sharing. Recent research on Grid has largely focused on issues of performance, scalability and standardization. Managing access to computing and data resources is a complex and time consuming task. As Grid computing matures, deciding which systems to use, where the data resides for a particular application domain, how to migrate the data to the point of computation (or vice versa), and data rates required to maintain a particular application "behavior" become significant.

Globus a middleware which is a de facto standard for grid computing. This project contains setup for a grid environment, in which 50s or 100s of execution nodes out of which one has been chosen as a submission node, and the other two are container nodes. These container nodes are actually having the schedulers of OpenPBS, SGE

and also condor which creates heterogeneous environment for clusters. When the submission node submits jobs to the containers, then these jobs are scheduled to run on other nodes attached to the OpenPBS and SGE clusters in parallel.

Agent and multi-agent technologies provide a promising approach to make Grid technologies and solutions based on Grid and Cluster technologies smarter, more flexible, and adaptable. Agents could play an important role in Grid Computing.

Traditional use of Grid Computing Systems has been for batch jobs in the scientific and academic computing. We envision the next generation Grid computing systems to support graphical interactive sessions.

1.1.1 Objective

Main objectives of project, are as follows-

- The first objective was to setup the grid environment , using the Globus Toolkit. Also setup and configured OpenPBS ,SGE ,and Condor schedulers on two of the grid nodes.
- Second objective was to design an Agent.
- Configuring heterogeneous grid environment.
- Using this Agents implement and integrate MultiAgent Framework for interactive Jobs.

1.1.2 Who can benefit from the Grid

- Medical/Healthcare
- Geological and climate applications
- Pharmaceutical, Chemical, Biotechnology
- Physics and Astrophysics

- Computer Science

1.1.3 Scope

The grid environment setup includes a network of 100s of workstation established in a lab of the university. One of the special features of this project is study and configuration of OpenPBS , SGE clusters and condor, which can certainly be helpful in applications.

As agent-level grids can take advantage of the capabilities of computational grids in supporting their load balancing and quality-of-service requirements. Also designing and implementing MultiAgent framework provides Interactive, and on-demand access to grid resources.

1.1.4 Thesis Organization

The Project work will be implemented in two major phases. The initial phase contains grid environment has been setup, which was the fundamental task. Next phase is to design, identify agent design, implement and integrate agents and job Management in heterogeneous framework. The rest of the thesis is organized as follows:

Chapter 2 includes literature survey.

Chapter 3 introduces grid computing and explains why it is needed and its types. It also explains the Globus Toolkit components, which are necessary for the functioning of the grid.

Chapter 4 includes the prerequisites, installation and configuration of the Globus on submission and container nodes.

Chapter 5 includes introduction of the interactive jobs, its requirements and its proposed architecture.

Chapter 6 introduces multiagent framework, what are agents, its model and agent design for project.

Chapter 7 contains job processing cycle, its components and resources needed to run a particular job.

Chapter 8 includes introduction to configuration of Condor on different nodes. It also explains how to connect it with globus and job execution in Condor.

Chapter 9 covers the whole implementation of agents which has been deployed on the grid. It includes all information about how jobs were taken, how the application has been designed and how it has been deployed on the grid. It also includes the time taken to execute the job.

Chapter 10 includes conclusion and future work that can be done on the grid.

Chapter 2

Literature Survey

The Grid and agent communities both develop concepts and mechanisms for open distributed systems, albeit from different perspectives. The Grid community has historically focused on brawn: infrastructure, tools, and applications for reliable and secure resource sharing within dynamic and geographically distributed virtual organizations. In contrast, the agents community has focused on brain: autonomous problem solvers that can act flexibly in uncertain and dynamic environments. Yet as the scale and ambition of both Grid and agent deployments increase, we see a convergence of interests, with agent systems requiring robust infrastructure and Grid systems requiring autonomous, flexible behavior [2]

Agent technology provides several concepts, which allow analysts to design applications in a way close to the human thought. Furthermore, agents provide applications with useful features for dealing with complex and dynamic environments. Agents are accepted as a powerful high-level abstraction for modelling complex software systems. It is intended that an entire system be built of a hierarchy of identical agents with the same functionality. Agents are considered both service providers and service requestors.

2.1 Methodology

- While many agents can be utilized as a compute intensive multi-agent system, the agents are not predetermined to work together. They possess their own motivation, resource and environment.
- Autonomy is used to describe the character of an agent, with regard to its intelligence and social ability. An agent can fulfil high-level tasks directly or through cooperation with other agents.
- Architecture is used to provide a framework for interaction between agents [3]. For example, multiple agents can be organized into a hierarchy.

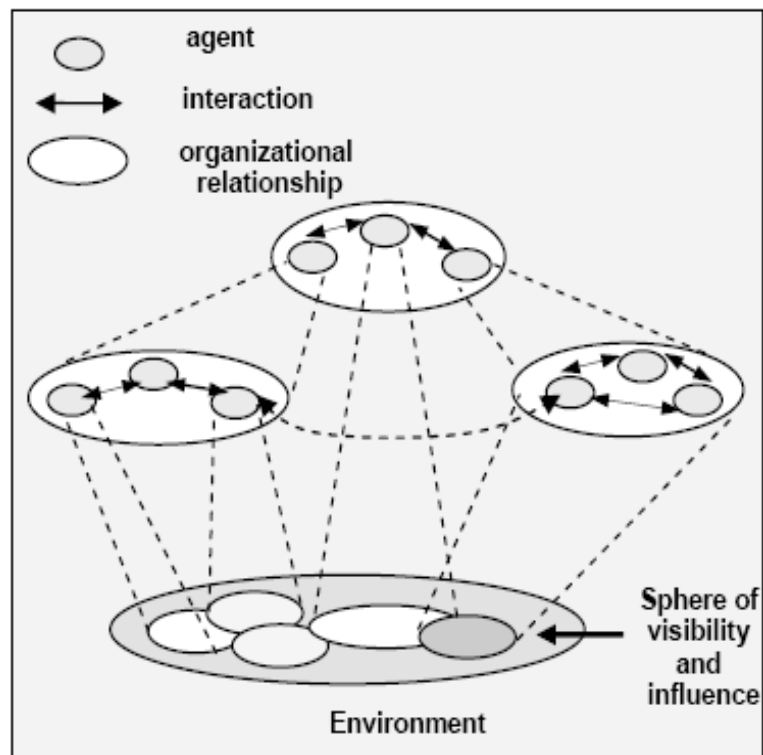


Figure 2.1: View of MultiAgent Framework

2.2 Batch and Interactive Jobs

Generally, there are two kinds of job submission methods: batch execution and interactive execution. When a batch job is submitted, it is submitted to a queue, where it waits until it reaches the top of the queue and the required resources become available. An interactive job [4] is a job where the nodes are reserved for users to log into and issue commands by hand. An interactive job is run immediately upon submitting if the specified resource are available. Since the interactive jobs never wait, and all the input and output are handled transparently, it is very useful for testing batch scripts and debugging programs. Although interactive jobs may decrease the utilization of the system, it is vital for a working scientific computing system.

Traditional use of Grid Computing Systems has been for batch jobs in the scientific and academic computing. We envision the next generation Grid computing systems to support graphical interactive sessions. In this project a resource management framework has been proposed for supporting graphical interactive sessions in a Grid computing system. The high level architectural resource management framework distributed among the submission node, central scheduler node, and the execution node.

2.3 Performance Analysis

Paper [5] describes performance measurement of two different agent based technologies. The performance measures evaluated were execution time and generated traffic. For each agent platform, average round trip times needed for the circular exchange of messages between two agents and the traffic generated during this process was measured. The motivation for this was to determine the maximum amount of user data that can be transferred over the network in a limited amount of time, as well as the message overhead. The results indicate that technology which uses Java to code user data, had the lowest RTT when sending large messages. This system uses java to

provide portability and flexibility while increasing whole system performance.

2.4 Section Summary

The Globus is widely used as middleware to successfully provide Grid services. However, previously it was impossible to submit interactive jobs to a Globus Gatekeeper. A novel way is introduced to submit interactive jobs to Grid resources via Grid computing. It is based on a GSI which uses user authentication. Interactive jobs are submitted to an interactive secure shell process rather than directly to a Globus Gatekeeper.

Chapter 3

Introduction to Grid

3.1 What is Grid

A grid is a system that

- o Coordinates resources that are not subject to centralized control Grid integrates and coordinates resources and users that exist within different control domains.
- o Uses standard open, general purpose protocols and interfaces : A grid is built from multipurpose protocols and interfaces that address such issues like authentication, authorization and resource discovery.
- o To deliver non-trivial qualities of services: A grid allows its constituent resources to be used in a coordinated fashion to provide various qualities of service like response time, throughput etc.

3.2 Types of GRID

- **Computational Grid:** This grid [6] is used to allocate resources specifically for computing power. The resources are usually high-performance machines.
- **Data Grid:** This grid is used for housing and providing access to data over multiple organizations.

- **Scavenging Grid:** It is one type of Computational grid which uses the unused resources for computational power. It steals CPU cycles when CPU is idle, so it is also called Cycle Scavenging Grid.

3.3 GRID Architecture and Core Services

Grid uses Globus as a middleware its architecture consists components as shown in figure 3.1

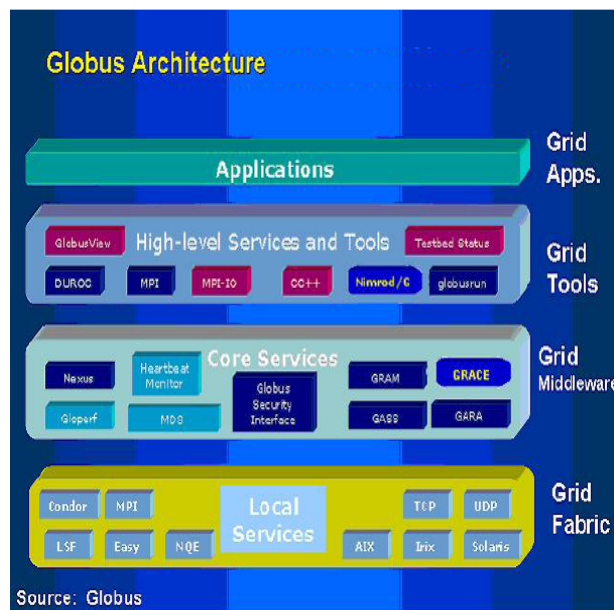


Figure 3.1: Globus Architecture

Main Grid services[7] includes:

- Scheduling (Globus Resource Allocation Manager)
- Information (Metacomputing Directory Service)
- Security (Globus Security Infrastructure)
- Health and Status (Heartbeat monitor)

- Remote file access (Global Access to Secondary Storage)
- Reservation of Resources in Advance (GARA)

3.3.1 GSI

A major requirement for grid computing is security. At the base of any grid environment, there must be mechanisms to provide security including authentication, authorization, data encryption, and so on. The Grid Security Infrastructure (GSI) component of the Globus Toolkit provides robust security mechanisms. The GSI includes an OpenSSL implementation. It also provides a single sign-on mechanism, so once a user is authenticated, a proxy certificate is created and used when performing actions within the grid.

As a very important component of GSI is the certificate. A certificate is used to identify all the users and the services present in the grid. Usually, there is a central Certificate Authority for a grid, which will issue certificates to each and every node present in the grid.

3.3.2 GRAM

GRAM is the module that provides the remote execution and status management of the execution. When a job is submitted by a client, the request is sent to the remote host and handled by the gatekeeper daemon located in the remote host. Then the gatekeeper creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates.

Gatekeeper

The gatekeeper daemon builds the secure communication between clients and servers. The gatekeeper daemon is similar to `inetd` daemon in terms of functionality. However, gatekeeper provides a secure communication. It communicates with the GRAM client

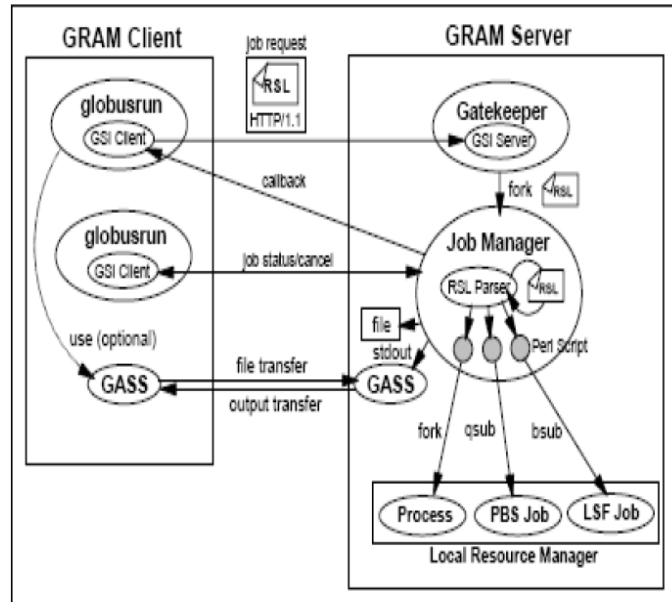


Figure 3.2: GRAM Architecture

and authenticates the right to submit jobs. After authentication, gatekeeper forks and creates a job manager delegating the authority to communicate with clients.

JobManager

Job manager is created by the gatekeeper daemon as part of the job requesting process. It provides the interfaces that control the allocation of each local resource manager, such as a job scheduler like PBS, LSF, or Condor.

3.3.3 MDS

MDS provides access to static and dynamic information of resources. Basically, it contains the following components:

O Grid Resource Information Service (GRIS): GRIS is the repository of local resource information derived from information providers. GRIS is able to register its information with a GIIS, but GRIS itself does not receive registration requests.

- o Grid Index Information Service (GIIS): GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIISs.
- o MDS client: A search for resource information that you want in your grid environment is initially performed by the MDS client.

3.3.4 Data Management

When building a grid, the most important asset within grid is data. Within design, user will have to determine their data requirements and how he will move data around infrastructure or otherwise access the required data in a secure and efficient manner. Standardizing on a set of grid protocols will allow to communicate between any data source that is available within design.

Globus provides the GridFTP and Global Access to Secondary Storage (GASS) data transfer utilities in the grid environment. In addition, a replica management capability is provided to help manage and access replicas of a data set.

Chapter 4

GLOBUS: Grid Preamble

Globus is a grid middleware, considered to be the de facto standard for grid computing. So, for practical experience of the grid computing it is best to start with the installation of Globus.

4.1 Configuration Details

- Operating System .. Fedora 4.0
- Middleware .. Globus 4.0
- Machines .. Machines having following configuration

nodeA	10.1.10.85	user-85a.nitdomain5.edu
nodeB	10.1.10.103	user-103a.nitdomain5.edu
nodeC	10.1.10.99	user-99a.nitdomain5.edu

4.2 Configuration Steps

The steps of GT4 [6] configuration are as follows:

4.2.1 Installing and Configuring Linux

- Installing and Configuring Linux
- Setting up accounts

4.2.2 Deploying Torque (OpenPBS)

Torque Open PBS or just PBS) is deployed on nodeB as a remote batch system, and then later configured Globus GRAM WS so that jobs can be submitted into PBS via Globus from the grid.

- a Deploying RSH
- b Downloading and Installing Torque
- c Configuring and Deploying Torque (PBS)
- d Starting PBS
- e Testing PBS

4.2.3 Deploying Sun Grid Engine (SGE)

Sun Grid Engine (SGE) is deployed on nodeC as a "remote" batch system, and then later configured Globus GRAM WS so that jobs can be submitted into SGE via Globus from the grid.

- a Downloading SGE
- b Unpacking the SGE distribution
- c Installing and Configuring SGE
- d Testing SGE

4.2.4 Deploying The PostgreSQL Relational Database

It is required to run the Globus Reliable File Transfer (RFT) service on nodes B and C since they are the head nodes for our “clusters”. RFT requires a relational database backend in order to preserve state across machine shutdowns.

a Preliminaries

b Initialization

c Starting the database

4.2.5 Fixing Java And ANT

The default java installed for Fedora Core 4 MUST be removed. It will not work with the Globus toolkit and it is easier and less troublesome to remove it entirely.

a Removing default Java

b Java installation

c Installing Ant

4.2.6 Deploying Globus

The Globus Toolkit should be installed as user 'globus' and not as root. The toolkit should be deployed on

- nodeA since that node will serve as the 'client' machine
- nodeB since that node will host the Globus GRAM WS that front ends the PBS batch system, along with other Globus grid services.
- nodeC since that node will host the Globus GRAM WS that front ends the SGE batch system, along with other Globus grid services.

- Preliminaries
- Building and Installing
- Installing updated packages
- Creating a Certificate Authority V. Obtaining a Host Certificate on nodeB
- Making a Copy for the Container
- Creating the grid-mapfile
- Configuring the RFT Service
- Configuring sudo
- Starting the Container
- Starting globus-gridftp-server
- Repeat for nodeA
- Obtaining Credentials for Generic User
- Testing the Grid Services on nodeB
- Obtaining host credentials for nodeA
- Testing globus-gridftp-server on nodeA
- Testing file staging
- Completing Deployment on nodeC

4.2.7 Connecting Globus GRAM WS and Torque (OpenPBS)

- I. Building the WS GRAM PBS jobmanager
- II. Testing the GRAM WS PBS jobmanager

4.2.8 Connecting Globus Gram WS and SUN Grid Engine

Now Globus GRAM WS on nodeC and SGE are connected so that jobs can be submitted into the SGE batch queue.

- a** Turning on reporting for SGE
- b** Building the WS GRAM SGE jobmanager
- c** Testing the GRAM WS SGE jobmanager

Chapter 5

Necessities for Interactivity

5.1 Introduction

Grid Computing technology provides resource sharing and resource virtualization to end-users, allowing for computational resources to be accessed as a utility. Resource Management is one of the key research areas for Grid Computing. Traditionally, Grid technologies have been used for executing batch jobs in the scientific and academic community. We believe that the application domains addressed by Grid technologies need to be extended to include graphical, interactive sessions. We propose interactive grids - next generation grids addressing the needs of graphical, interactive sessions. Interactive Grids permit end-users to access and control a remote resource eg. remote workstation in the Grid for graphical, interactive use.

Users want to use HPC systems in a manner similar to how they interact with their personal computer. Driven by these requirements, it is increasingly being recognized that a large pool of HPC users requires interactive, on-demand access to HPC resources. Interactive access is defined to be users can input information into the application and received timely visual output from the application, while on-demand access is defined as being allocated immediate access to the requested resources rather than having to wait for resources to be allocated by a batch queuing system. Pro-

viding users with interactive, on-demand access to an HPC system will afford these users:

- Quicker first-time access to the system.
- Faster code debug cycles.
- Having to their jobs according to their needs.

There are number of challenges to implementing interactive, on-demand access to HPC systems including:

- Since HPC resources still cost nowhere near a level where every potential user can have their personal HPC resource in their office or cubicle, how can the sharing of the interactive, on-demand HPC resources be abstracted in such a way the users feel like they are not sharing the resources with many other users.
- Given the number of potential users for an HPC system, the number and duration of application executions, and the number of simultaneous processors that each application execution requires, how many processors and other resources will be required?
- How can broader interactive, on-demand access enable greater visual feedback and visualization of data being processed?
- Are there interactive, on-demand access policies that are fair to all of the users of such an HPC system?

5.2 Requirements

We are considering interactive grids - Grid computing systems that extend the application domain to include graphical interactive sessions. Specifically, an Interactive Grid Computing System allows the end-user access and control of a remote resource

in the Grid for graphical, interactive use. To enable such grids, we require a resource management architecture that effectively manages the vast heterogeneous resources across administrative domains, as well as effectively manages the resources during the graphical interactive session.

5.3 Proposed Architecture

The important issues to consider for a resource management framework for interactive grids [8], as compared to traditional batch-oriented grids are:

- Providing QoS guarantees for graphical sessions.
- Guaranteeing SLAs per graphical session.
- Accurate prediction of application behavior and resource load.

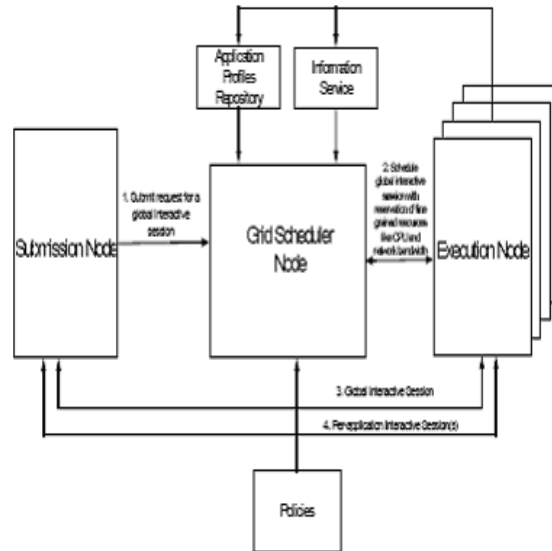


Figure 5.1: Overview Of Interactive Grid Computing System

Interactive Grid Computing System: An interactive grid computing system is a Grid computing system supporting Graphical Interactive sessions to remote

nodes. At a high level, it consists of Submission nodes, a Distributed Resource Management System, Execution nodes and Storage nodes. The end-user submits job requests through a submission node, and is given access to a remote execution node for graphical, interactive use.

oGlobal Interactive session: A global interactive session constitutes the association between the end-user and the remote execution node, wherein: the end-user interacts with the remote execution node to launch one or more applications, and subsequently interacts with the launched applications through per-application sessions.

oPer-application interactive session: A per-application interactive session for an application executing on the remote execution node, constitutes the association between the end user and the executing application, wherein: the end-user interacts directly with the application. A per-application interactive session occurs in the context of a global interactive session. We are most interested in graphics application sessions. However, our proposed solution would also work with text only applications as a special case.

Application Profiles: The application profiles contain the estimated CPU and bandwidth required for various classes of applications to provide acceptable frame rate and performance levels while executing remotely in an Interactive Grid computing system.

Chapter 6

MultiAgent Framework for Job Management

6.1 Agent

An agent is a software entity that shows several degrees of autonomy, since it has to take decisions and to carry out jobs without the direct participation of the user. Often an agent is an active object, i.e., an object with autonomous computational capability. A mobile agent is an autonomous software entity, which has the ability to roam among the nodes in a network-aware fashion. Some of the advantages of agents are:-

- Lower consumption of bandwidth
- Tolerate unreliable connections, i.e., fault-tolerance
- Can be used to balance the load among the connected nodes.
- Supports very flexible design

6.1.1 Working of An Agent

All components and sub-systems in a grid environment can be represented as agents, and each agent is registered with a local or system-wide manager called facilitator. The whole system is divided into two layers for managing different tasks, i.e., local management and system-wide management.

In local management, general-purpose agents are placed on network devices to interact with the resources, which are distributed, and manage the local events, thus reducing the workloads for the local manager. To represent local resources, each general-purpose agent collects relevant identification information, resource information, state information, etc.

A facilitator agent is defined as a server agent. Normally, these facilitators are located at tree nodes or hosts on the network topology, where each tree node corresponds to a specific subnet and each subnet consists of several host.

A user interface agent is responsible for receiving inputs, sending requests to the facilitator for delegation of tasks to appropriate agents, and displaying the results.

The other two categories of agents are application agents and meta-agents. An application agent is responsible for providing a collection of services of a particular type. These services could be domain independent, user-specific, or domain-specific. While a facilitator processes domain-independent coordination strategies, a meta-agent is used to assist the facilitator agent in coordinating the activities of other agents by using domain- and application specific knowledge or reasoning. Inter-agent Communication Language (ICL) is to set up the communication between agents and facilitator.

6.2 Agent Based Grid Design

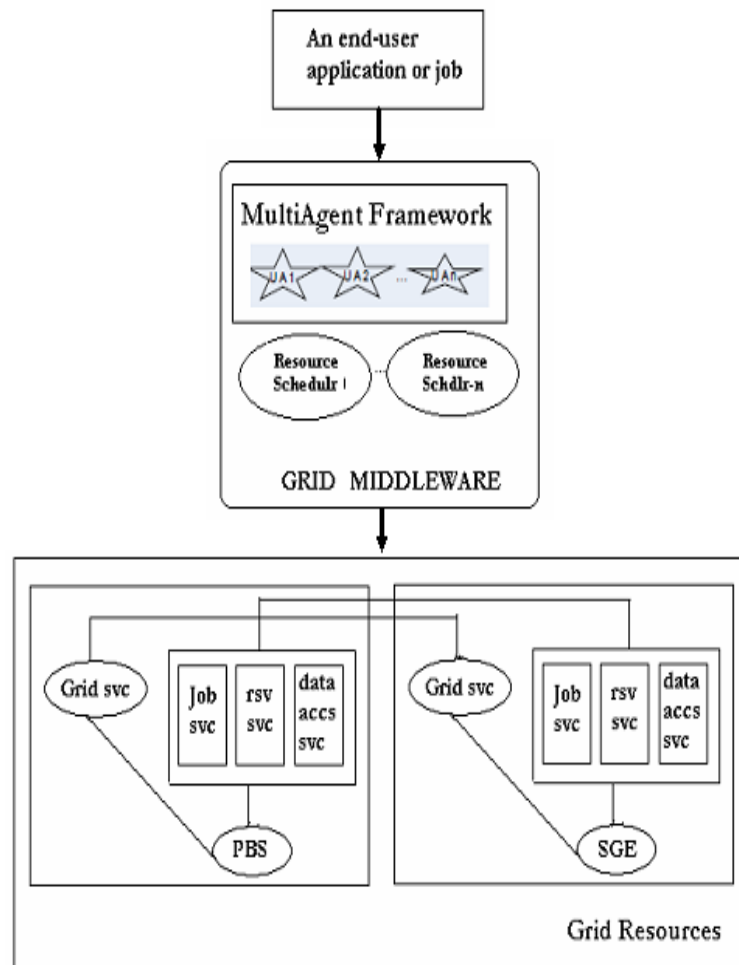


Figure 6.1: Agent Based Grid System hierarchy

The multi-agent system-based hierarchy grid architecture model highly abstracts the substance application. It can be divided into layers in logic, as shown in Figure 6.1.

6.3 MultiAgent Framework Design

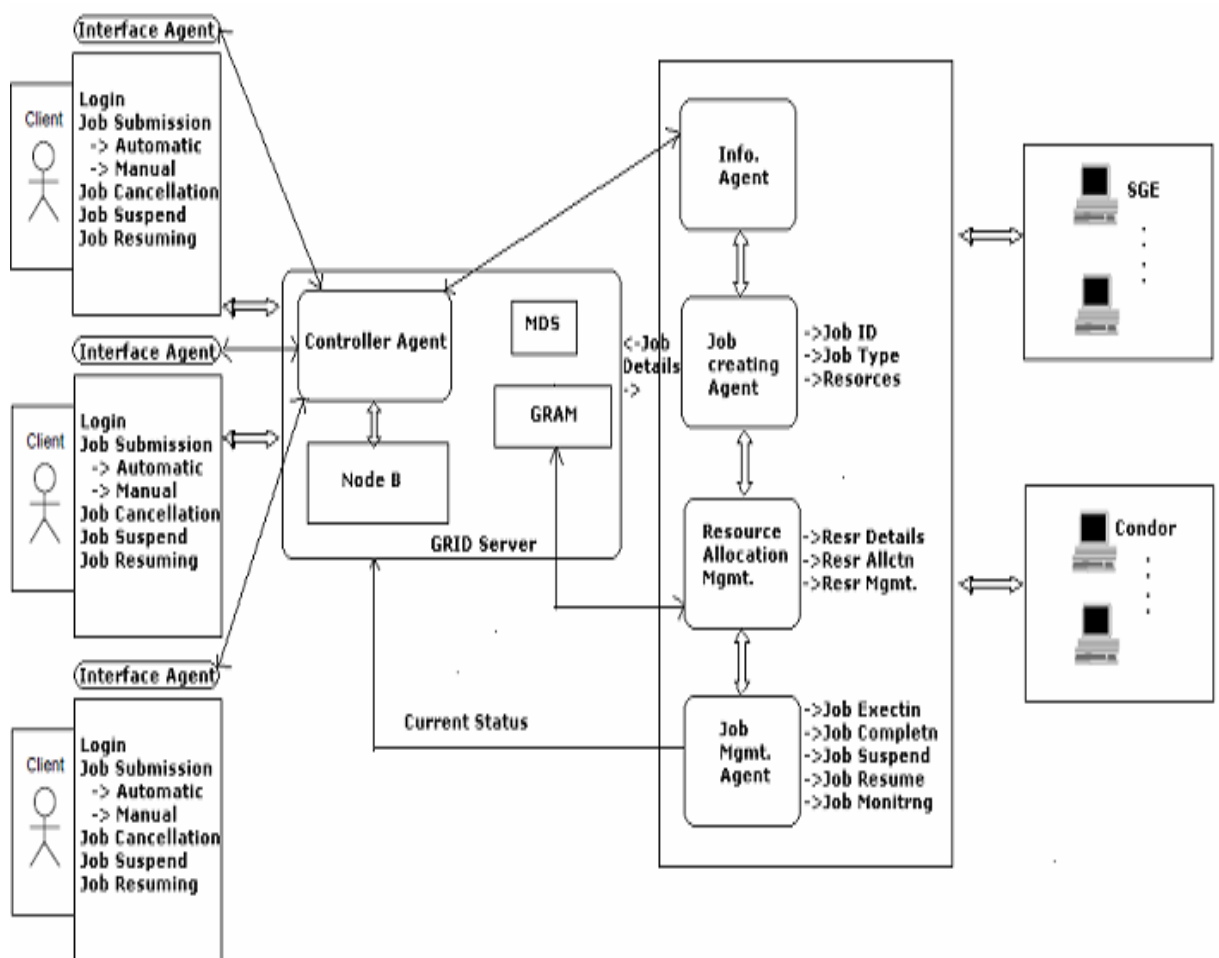


Figure 6.2: MultiAgent Design

Figure 6.2 describes how work process is done in multiagent framework. Work process is presented in following steps:

- Through Interface Agent user submits a job.

- Each and every Agent communicates with controller agent which is the master agent.
- Controller Agent through GRAM calls Info and Resource Allocation Agent to assign a suitable node and resources.
- Resource database returns nodes itinerary to the Info Agent.
- Job creating Agent assigns job-id and resources.
- Each Agent begins a user process on an available node.
- After the available node execution, the result is back to the sub-agents.
- Sub-agents pass the result to their main master Agent or Controller Agent.
- Controller Agent handles with all the back results and submits the ultimate result to the user thorough Interface Agent.

Chapter 7

Job processing Methodology

Grid computing allows sharing of resources from different administrative bodies to end users. Execution of programs submitted by arbitrary users is an important feature of grid system. This resource sharing can compromise the security of a resource, affecting the integrity of the resource. The integrity of the resource machine is important as crashes due to malicious actions from a foreign job can disrupt many other executing jobs. Unnecessary resource down time incurs lost of precious time for job execution.

There are four main entities in architecture, which are clients, middleware, schedulers and resources. A client is a user who submits a job to the system. A job refers to a collection of Java classes that the client wants to execute. The job is submitted by the client to the web portal through a graphical user interface (GUI). Also agent is an entity which helps to achieve this. The agent delegates the management of jobs to schedulers. A scheduler divides a job into smaller tasks (in the case of an independent job, a task refers to the subset of parameters that can be executed independently) and sends the tasks to the resources for execution. Figure shows the proposed architecture model.

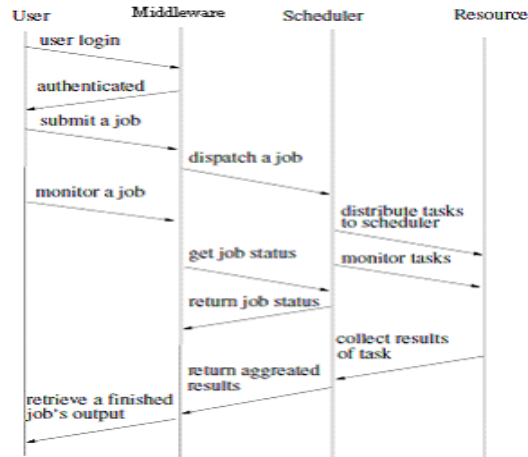


Figure 7.1: job processing sequence

7.1 Job definition

The clients submit a job to the server in the form of Java class files together with the job descriptions and the input parameters for the Java class files. The job descriptions are stored in a separate file in XML format. This file is submitted together with the Java class files during the job submission. The job descriptions consist of:

- o Job Priority: This information is required by the scheduler to be able to manage jobs optimally. The scheduler can charge different price for scheduling different priority job.
- o Job Scheduling policy: The policy is used by the scheduler to choose suitable resources for the job execution. These policies lead to a scheduling decision for speed, cost or other criteria of optimizations. The optimizations determine the cost of job execution charged by the resources.
- o Job Requirements: This information is used by the scheduler to select the resources for the job execution. The requirements cover memory requirement, storage requirement, execution deadline time. Use of this information allows the scheduler to decide whether it requires extra resources.

o Job execution time: Job's execution time. This information is required by the resource to check whether a job is working correctly. Thus if a job is running for unreasonably longer period than the approximated execution time, the resource machine can decide to terminate the job and return an error message to the job owner.

7.2 Job Execution and States

Job Execution takes following steps:

- Event tells the server to start scheduling cycle.
- Server sends scheduling command to scheduler.
- Scheduler requests resource info.
- Server returns the requested info.
- Scheduler request job info from server.
- Server sends job status info to scheduler and scheduler makes the policy decision to run the job.
- Scheduler sends run request to server.
- Server sends job to run.

Figure 7.2 how grid services deals and manages with submitted job.

7.3 Job Management

The basic object that grid manages is the job. So in the job has the following properties: o Batch or Interactive

- o Defines a list of required resources
- o Defines a priority
- o Defines the time of execution

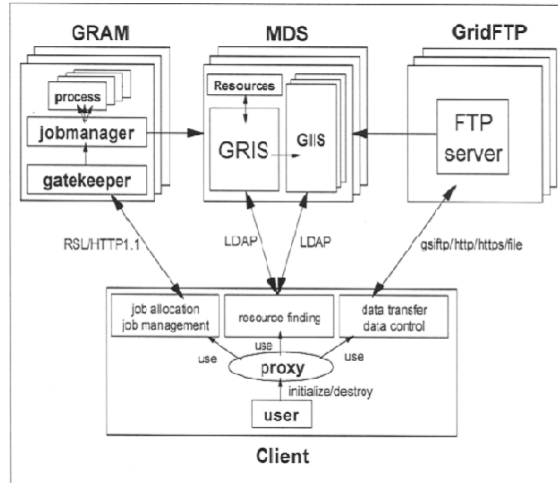


Figure 7.2: Job execution using grid services

7.4 Resources

Each submitted job can specify a list of resources required for its execution: the number and the type of the specifiable resources is platform dependent.

- host: name of the host on which job should be run.
- nodes: number and/or type of nodes to be reserved for exclusive use by the job.
- number of CPUs: number of CPUs requested by job

7.5 Performance parameters

- Resource Discovery : Time taken to find resources for the particular job.
- Selection of best resource : Amount of time taken by a agent to select a best resource.
- Job Execution Time : Total time taken to execute a particular job.
- job interarrival times are assumed to be independent and identically distributed.

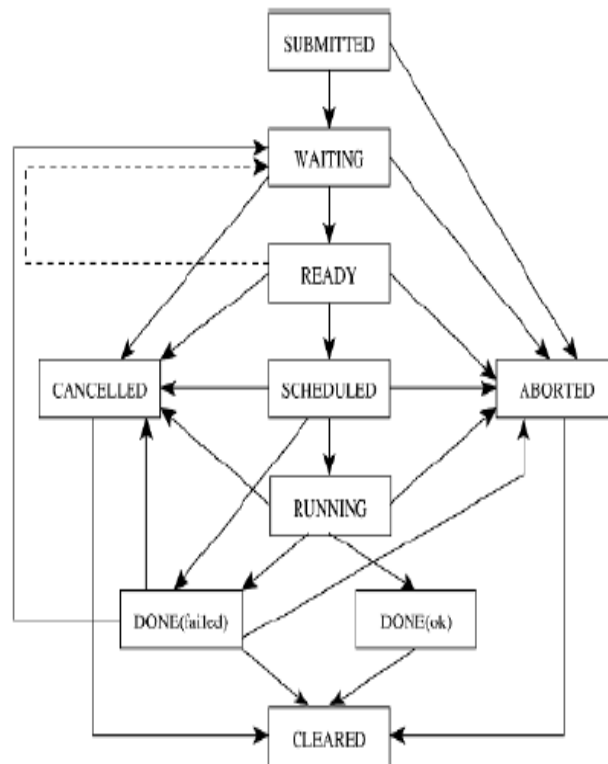


Figure 7.3: Job Execution Sequence

- Size of the input data processed by a job is proportional to that jobs length.
- Average RTT: Shows the time needed for the circular exchange of messages between agents.
- Data Transferred: Shows the data transferred by agents during the exchange of messages.
- Message Overhead: Shows the traffic generated by the agent platform in order to transfer the actual data.

Chapter 8

Eneration of Supercomputing Power

After configuring Globus, the Grid is ready to accept the jobs. The next task is to generate the supercomputing power i.e. large computational capability by means of connecting several machines to the grid and utilizing their resources for computation.

A goal of grid computing is to allow the utilization of resources that span many administrative domains. A Condor pool often includes resources owned and controlled by many different people. Yet collaborating researchers from different organizations may not find it feasible to combine all of their computers into a single, large Condor pool. Condor shines in grid computing, continuing to evolve with the field.

Condor can be used to manage a cluster of dedicated compute nodes. In addition, unique mechanisms enable Condor to effectively harness wasted CPU power from otherwise idle desktop workstations. For instance, Condor can be configured to only use desktop machines where the keyboard and mouse are idle. Should Condor detect that a machine is no longer available (such as a key press detected), in many circumstances Condor is able to transparently produce a checkpoint and migrate a job to a different machine which would otherwise be idle. Condor does not require a shared file system across machines - if no shared file system is available, Condor can transfer

the job's data files on behalf of the user, or Condor may be able to transparently redirect all the job's I/O requests back to the submit machine. As a result, Condor can be used to seamlessly combine all of an organization's computational power into one resource.

Condor can be used to build Grid-style computing environments that cross administrative boundaries. Condor's "flocking" technology allows multiple Condor compute installations to work together. Condor incorporates many of the emerging Grid-based computing methodologies and protocols. For instance, Condor-G is fully interoperable with resources managed by Globus.

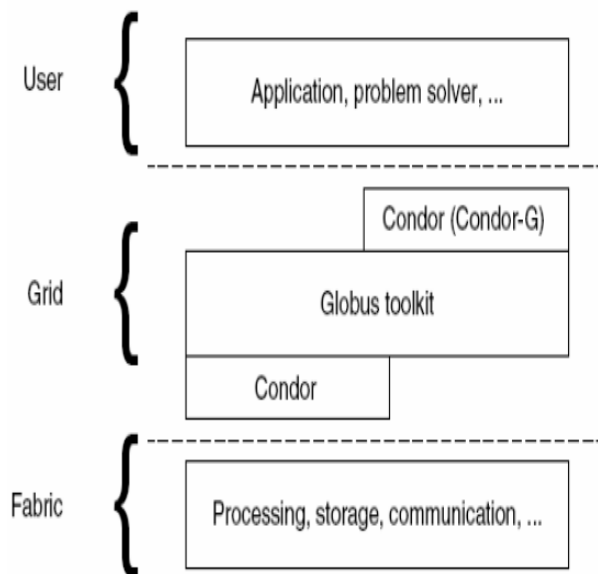


Figure 8.1: Condor Architecture Combined with Grid

Figure 8.1 describes the association of condor with the grid platform. The upper most layer is consist of Users. They are having applications, problem solving programs or softwares etc.

The middle layer is comprised of Grid platform. The condor is associated with grid platform here. Condor-G is a special variant of condor made for Globus. It works at both side of the grid platform. It has intermediate interface with user and

grid platform as well as interface between grid platform and the grid fabric. Condor provides functionality of allowing multiple nodes to be connected to grid platform with ease. It also contains some security aspects which are not concerned for this project.

8.1 Installation and Configuration

8.1.1 Installation

The first step is to download condor package from the website of University of Wisconsin - <http://www.cs.wisc.edu/condor>. It can be downloaded as binary, source or rpm.

Assuming that binary package is downloaded for Linux platform do `rpm -ivh packagename` on the terminal.

Configuration of central-manager and submitter-host

The following is a list of steps which were required in order to get a condor cluster up and running. All machines in the (initial) cluster were wiped and had a fresh install of Fedora Core 4. However, it shouldn't be too much of a stretch to get this working on any FC4 machine. Throughout, I used the precompiled binary rpm versions of condor. Unless stated, these steps are the same for the head node and the slave nodes.

- Get the latest stable version of condor from <http://www.cs.wisc.edu/condor/downloads>
I got the rpms for version 7.0.5.
- Next need to install a C++ library for backwards compatibility. This may not be needed if the condor binaries have been recompiled recently or you already have this library installed. I used yum, with the default repos as follows:
`yum install compat-libstdc++-33-3.2.3-47.fc4.i386`.
- Create a new user named condor (as root).

- Condor has been installed into: /opt/condor

Globus will use Condor as a scheduler on nodeC and so Globus GRAM should be able to submit jobs to Condor. Hence, Condor should be configured under the ownership of globus user who is the owner of Globus container too.

To configure Condor as central-manager and submitter-host run the following file in Condor-home from terminal with globus user as owner:

Condor_configure --owner=globus --type=submit,manager

Next, the condorconfigure file which keeps all the configuration settings should be modified for condor usage. Edit this file stored in etc directory in Condor home. Make changes in this file mentioned below:

Set RESERVED_SWAP=0. This will not block the job from being started even though swap memory can not be reserved before job is started.

Set HOSTALLOW_READ=* and HOSTALLOW_WRITE=* so that no node will be blocked from writing the output files back once the jobs are completed.

Set JAVA=installation_dir_of_jdk to let condor work with java properly.

In order for Condor to work properly you must set your CONDOR_CONFIG environment variable to point to your configuration file. Set the environment by adding following line in the .bash_profile file

- export CONDOR_CONFIG=(Condor_installation_dir)/etc/condor_config
- export PATH=\$PATH:(Condor_installation_dir)/bin
- export PATH=\$PATH:(Condor_installation_dir)/sbin

8.2 Adding Execution Hosts

Condor is a heterogeneous execution system. Therefore, it can have either Windows or Linux nodes as execution nodes. For this project, Linux nodes were already used in the OpenPBS scheduling system. So many Windows machines were setup as condor execution nodes.

8.2.1 Installation of Condor client on Windows machine

First, download the Condor setup for Windows from the same website that is mentioned above for Linux Condor download. Run the setup on Windows node to be configured as execution node. Following steps are to be followed:

- First, enter the fully qualified host name or IP of nodeC when asked to join an existing Condor pool. This step suggests the need of nodeC to be setup as submitter and central manager node.
- Next since the nodes are simple execution nodes they need not submit jobs into Condor pool.
- Whether the jobs submitted to this node should always run or they should be suspended at some local activity is set.
- Then setup asks DNS name or IP.
- User has to enter path to java. It is advisable to have same version of jdk installed on each Windows host as that was installed on nodes A, B and C.
- Setup prompts next for hosts access on this node. Set both read and write access to * so that no host will be blocked because of that.
- User can accept rest of the options as default ones and install Condor.
- Once Condor has been installed, the condor_master daemon will be started as a service everytime on a start-up.
- Change the host file in c:/windows/system32/drivers/etc/host file and make an entry of central manager

8.3 Connecting Globus GRAM-WS And Condor

Condor is a scheduler system which Globus toolkit uses to schedule jobs. Even though, both Globus toolkit and Condor have been installed, Globus toolkit can not use Condor until Condor is connected with Globus GRAM-WS.

8.3.1 Setup Globus job manager for Condor

Environment has to be setup in first place. Edit `.bash_profile` file stored in home directory of globus user in nodeC and environment variables for `containercert.pem` and `containerkey.pem`:

- `export X509_USER_CERT=$GLOBUS_LOCATION/etc/containercert.pem`
- `export X509_USER_KEY=$GLOBUS_LOCATION/etc/containerkey.pem`
- Make sure `(Condor_home)/bin` and `(Condor_home)/sbin` are in `PATH`.
- As globus user, change to the directory `gt-4.0.1-all-source-installer` in home of globus user.
- Now, the package `gt4-gram-condor` is to be built.
- **make gt4-gram-condor**
- Install the built package by:**make install**
- Configure the job-manager by running following:
`$GLOBUS_LOCATION/setup/globus/setup-globus`
`-job-manager-condor`

8.4 Job manager for Condor

The setup changes the Globus container. After the setup is run, the container also spawns `GRAM-scheduler-event-generator`. This event-generator keeps track of the

current status of the job by referring to the log files generated by Condor and forwards the job status to the GRAM.

This setup also generates a Perl script named `condor.pm`. When GRAM schedules a job to run on Condor, this script generates a job submission script based on job description provided by user. This job submission script is recognized by Condor. So GRAM submits the job submission script to Condor queue and the job is scheduled.

8.5 Configuring Globus to submit to a homogeneous Condor pool

- 1 Make sure `$GLOBUS_LOCATION` and/or `GPT_LOCATION` are set appropriately.
- 2 Run the following command: `$GLOBUS_LOCATION/setup/globus/setup-globus -gram-job-manager -type=condor`
- 3 Edit `$GLOBUS_LOCATION/etc/grid-services/jobmanager-condor` to add the parameters `-condor-arch` and `-condor-os`, which specify what ARCH/OPSYS to use. The results should look something like this (everything on a single line), except that will change the pathnames to be correct for your system:


```
stderr_log,local_cred - /scratch/globus/libexec/globus-job-manager globus
-job-manager -conf /scratch/globus/etc/globus-job-manager.conf -type
condor -rdn jobmanager-condor -machine-type unknown -condor-arch IN-
TEL -condor-os LINUX
```
- 4 Run the following command: `$GLOBUS_LOCATION/setup/globus/setup-globus -gram-reporter -type=condor`

8.6 Configuring Globus to submit to a heterogeneous Condor pool

- 1 Make sure `$GLOBUS_LOCATION` and/or `GPT_LOCATION` are set appropriately.

- 2 Change the Globus etc directory: `cd $GLOBUS_LOCATION/etc.`

Now, perform the following steps once for each platform in pool:

- 3 Make a copy of the `globus-job-manager.conf` file (the new file will be specific to this particular platform):

```
cp globus-job-manager.conf globus-job-manager-condor-INTEL-LINUX.conf
```

- 4 Edit the `-globus-host-cputype/manufacturer/osname/osversion` options in the new conf file to match those of the platform.

- 5 Make a copy of the `grid-services/jobmanager` file (the new file will be specific to this particular platform). The name of this file will be the name that user will use to submit jobs to this platform:

```
cp grid-services/jobmanager grid-services/jobmanager-condor-INTEL-LINUX
```

- 6 Edit the new file in `grid-services` to change the `-conf`, `-type`, and `-rdn` parameters and add the `-condor-arch` and `-condor-os` parameters (which specify which ARCH/OPSYS to use). The end result should look something like this (all on one line), except that you will change the pathnames to be correct for your system::

```
stderr_log,local_cred - /scratch/globus/libexec/globus-job-manager globus
-job-manager -conf /scratch/globus/etc/globus-job-manager-INTEL-
LINUX.conf -type condor -rdn jobmanager-condor -machine-type un-
known -condor-arch INTEL -condor-os LINUX
```

8.7 Testing Setup

To test Condor as a stand alone system, start `condor_master` on central manager nodeC by running `condor_master` command as `globus` user.

Then run `condor_on` command so that `condor_master` will spawn all other necessary daemons on the central manager like `condor_schedd`, `condor_startd`, `condor_master`, `condor_collector`, `condor_negotiator`.

8.8 Generating SuperComputing Power

To generate supercomputing power 100s of pcs has been integrated with grid. Parameters like GFlops, Load Average, Condor Load, Activity, Hostname of all the machines generating heterogeneous grid environment has been taken so that their aggregation power of all the machines help to achieve this goal. By integrating and viewing and fetching all the required parameters results is as given in figure 8.2.

Figure 8.2 shows all the parameters with the GFlops. Hostname in the figure are workstation from different labs in Nirma University.

NIRMA Grid								
Name	Machine	Condor Load Ave...	Load Average	Operating System	KFlops	Mips	Total Load Avera...	Total Condor Lo...
slot1@A104-CC...	A104-CC-PC-61	0.000000	0.000000	WINNT51	606061	1585	0.210000	0.000000
slot2@A104-CC...	A104-CC-PC-61	0.000000	0.210000	WINNT51	606061	1585	0.210000	0.000000
slot1@A104cc-u...	A104cc-user54	0.000000	0.000000	WINNT51	660574	2018	0.000000	0.000000
slot2@A104cc-u...	A104cc-user54	0.000000	0.000000	WINNT51	660574	2018	0.000000	0.000000
slot1@A104cc-u...	A104cc-user55	0.000000	0.000000	WINNT51	675521	2148	0.080000	0.000000
slot2@A104cc-u...	A104cc-user55	0.000000	0.080000	WINNT51	675521	2148	0.080000	0.000000
slot1@A104cc-u...	A104cc-user69	0.000000	0.070000	WINNT51	636392	2025	0.070000	0.000000
slot2@A104cc-u...	A104cc-user69	0.000000	0.000000	WINNT51	636392	2025	0.070000	0.000000
slot1@A104cc-u...	A104cc-user70	0.000000	0.000000	WINNT51	632872	2148	0.070000	0.000000
slot2@A104cc-u...	A104cc-user70	0.000000	0.070000	WINNT51	632872	2148	0.070000	0.000000
slot1@A104cc-u...	A104cc-user72	0.000000	0.190000	WINNT51	665375	2018	0.190000	0.000000
slot2@A104cc-u...	A104cc-user72	0.000000	0.000000	WINNT51	665375	2018	0.190000	0.000000
slot1@b201-1-...	b201-1-11.nitd...	0.000000	0.480000	WINNT51	627953	2148	0.480000	0.000000
slot2@b201-1-...	b201-1-11.nitd...	0.000000	0.000000	WINNT51	627953	2148	0.480000	0.000000
slot1@b201-1-...	b201-1-13.nitd...	0.000000	0.010000	WINNT51	636392	2140	0.010000	0.000000
slot2@b201-1-...	b201-1-13.nitd...	0.000000	0.000000	WINNT51	636392	2140	0.010000	0.000000
slot1@b201-1-...	b201-1-14.nitd...	0.000000	0.000000	WINNT51	661913	2140	0.010000	0.000000
slot2@b201-1-...	b201-1-14.nitd...	0.000000	0.010000	WINNT51	661913	2140	0.010000	0.000000
slot1@b201-1-...	b201-1-19.nitd...	0.000000	0.010000	WINNT51	627831	2023	0.010000	0.000000
slot2@b201-1-...	b201-1-19.nitd...	0.000000	0.000000	WINNT51	627831	2023	0.010000	0.000000
slot1@b201-1-...	b201-1-20.nitd...	0.000000	0.020000	WINNT51	670900	2140	0.020000	0.000000
slot2@b201-1-...	b201-1-20.nitd...	0.000000	0.000000	WINNT51	670900	2140	0.020000	0.000000
slot1@b201-1-...	b201-1-21.nitd...	0.000000	0.000000	WINNT51	601811	1735	0.030000	0.000000
slot2@b201-1-...	b201-1-21.nitd...	0.000000	0.030000	WINNT51	601811	1735	0.030000	0.000000
slot1@b201-1-...	b201-1-27.nitd...	0.000000	0.000000	WINNT51	645847	2025	0.010000	0.000000
slot2@b201-1-...	b201-1-27.nitd...	0.000000	0.010000	WINNT51	645847	2025	0.010000	0.000000
slot1@b201-1-...	b201-1-28.nitd...	0.000000	0.000000	WINNT51	465537	2025	0.010000	0.000000
slot2@b201-1-...	b201-1-28.nitd...	0.000000	0.010000	WINNT51	465537	2025	0.010000	0.000000
slot1@b201-1-...	b201-1-29.nitd...	0.000000	0.010000	WINNT51	651177	2025	0.010000	0.000000
slot2@b201-1-...	b201-1-29.nitd...	0.000000	0.000000	WINNT51	651177	2025	0.010000	0.000000
slot1@b201-1-...	b201-1-30.nitd...	0.000000	0.000000	WINNT51	660574	2148	0.010000	0.000000
slot2@b201-1-...	b201-1-30.nitd...	0.000000	0.010000	WINNT51	660574	2148	0.010000	0.000000
slot1@b201-1-...	b201-1-36.nitd...	0.000000	0.000000	WINNT51	651487	2018	0.010000	0.000000
slot2@b201-1-...	b201-1-36.nitd...	0.000000	0.010000	WINNT51	651487	2018	0.010000	0.000000
slot1@b201-1-...	b201-1-38.nitd...	0.000000	0.000000	WINNT51	660394	1916	0.030000	0.000000
slot2@b201-1-...	b201-1-38.nitd...	0.000000	0.030000	WINNT51	660394	1916	0.030000	0.000000
slot1@b201-1-...	b201-1-4.nitdo...	0.000000	0.000000	WINNT51	677246	2025	0.000000	0.000000
slot2@b201-1-...	b201-1-4.nitdo...	0.000000	0.000000	WINNT51	677246	2025	0.000000	0.000000
slot1@b201-1-...	b201-1-43.nitd...	0.000000	0.030000	WINNT51	641919	2023	0.030000	0.000000
slot2@b201-1-...	b201-1-43.nitd...	0.000000	0.000000	WINNT51	641919	2023	0.030000	0.000000
slot1@b201-1-...	b201-1-44.nitd...	0.000000	0.000000	WINNT51	646883	2140	0.000000	0.000000
		0.0	1.2900000000...		26809318	85226	2.5799999999...	0.0

Figure 8.2: Generation of Super Computing Power

Chapter 9

Agent Implementation

9.1 Implementation Details

Generally there are two kinds of job submission methods, which are batch execution and interactive execution.

Agents can provide a useful abstraction at each of the Grid layers. By their ability to adapt to the prevailing circumstances, agents will provide services that are very dynamic and robust, and suitable for a Grid environment. Agents can be used to extend existing computational infrastructures. Few research groups have focused on offering an environment to combine the concept of computational grid and agents.

An agent is the essential component of this work. An agent can be depicted as an autonomous entity packaged of a set of internal modules, three of which, i.e., for internal scheduling, problem solving and social communication routing, are normative and others are optional. An agent has three basic capabilities Reactivity, Autonomy and Social Communication Ability.

- 1) Reactivity
- 2) Autonomy
- 3) Social communication ability

Normally a system is made of different agents, involving their distributed data,

knowledge, or control. A multi-agent system is composed of a collection of, possibly heterogeneous, autonomous entities which have their own problem-solving capabilities and interact with one another in order to achieve an overall goal. Multi-agent systems emphasize both the autonomy of individual agents and the cooperation between agents. Coordination is important for multi-agent [9] systems. In general, coordination is about using a mechanism to manage the interdependencies between the collaborative activities of agents.

Thus an agent is an object which works independently or autonomously to handle the tasks. For this paper the proposed agent will combine a few functionalities described above and work as an independent module for the client application. An agent will accept job parameters from the client and accordingly provide them to grid middleware for further measures and also keep the track of the job.

9.2 Agent Based Job Management

Job management is again an indispensable task for the grid computing environment. Execution of job requires the resources but before that it is necessary to identify the requirement of the resources by a particular job. Then after availability and allocation of the resources is next step. Monitoring the execution is also an important task and finally resources have to be made free for supplementary jobs. All these tasks have to be taken care by the job management system or JMS. Few existing tools functioning JMS are Condor, Torque and SGE. JMS usually consists of three components:

- a. Queue Manager
- b. Scheduler
- c. Resource Manager

The term interactive job management refers to availability and accessibility of various tasks of job while it is executing. We have identified following.

- a. Job Submission
- b. Job Status
- c. Job Suspending
- d. Job Resuming
- e. Job Destroy

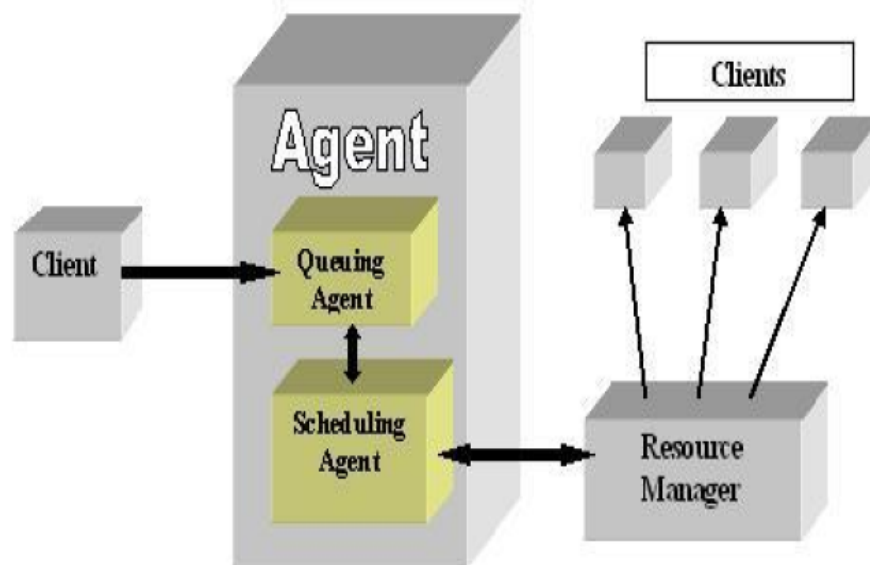


Figure 9.1: Job Management System Architecture

9.3 Agent Program Flow

The flow of the work would be as described in the figure 9.2 An initiating action would be triggered by the client. The client will submit the job through an interactive console to the system. After that an agent, which would be invoked by submitting a job to the grid environment, will take care of rest of the events. The agent will accept the job from client and will identify required parameters from the job.

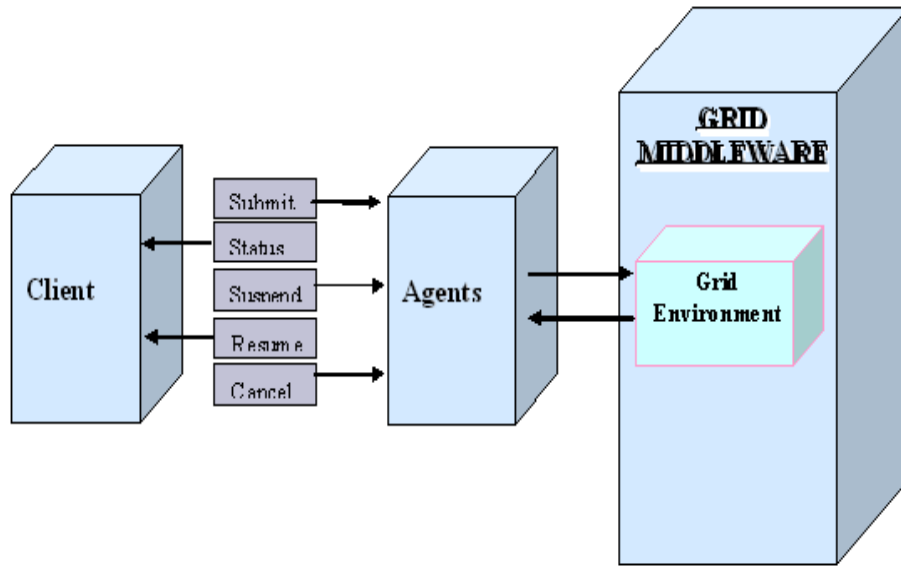


Figure 9.2: Representation of Agent Based System

Agent will then submit those to the grid middleware for identifying required resources and allocating them. Meanwhile for the executing job the agent will reside there in the environment as an active object until job is finished (or cancelled or destroyed).

9.4 MultiAgent Programming

In computational grids, performance based application applications need to simultaneously tap the computational power of multiple, dynamically.grid programming environments (a) need to be portable to run on as many sites as possible, (b) they need to be flexible to cope with different network protocols and dynamically changing groups of compute nodes.

Many researchers believe that Java will be a useful technology to reduce the complexity of grid application programming.Hence in this system java has been used to provide both the functionalities and platform independency [10].

9.4.1 Working of an MultiAgent System

Various java classes has been implemented to perform operations like login, file browsing , choosing job option, showing the status, terminating etc. This classes are as follows:

I Login authentication is performed in this class. Login operation is defined and user interaction to input username and password is described shown in 9.3and welcomes user.

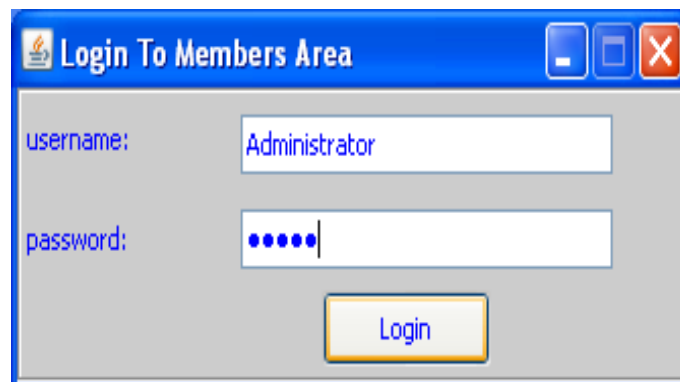


Figure 9.3: UserLogin

- II** CreateProxy class Creates a proxy before submitting any job using grid-proxy-init shown in 9.6.
- III** In JobSubmit class Job Submission is done according to user's need using this class. When a job is submitted it will give a message to user that job has been submitted successfully shown in 9.7.
- IV** Class Checkpoint suspends and resumes a job modifying a cluster queue. In this system cluster of PBS and SGE has been used. Using the queue of a particular cluster user can change the the state of the job and restart later at the point from where it was suspended.
- First compile a job want to submit

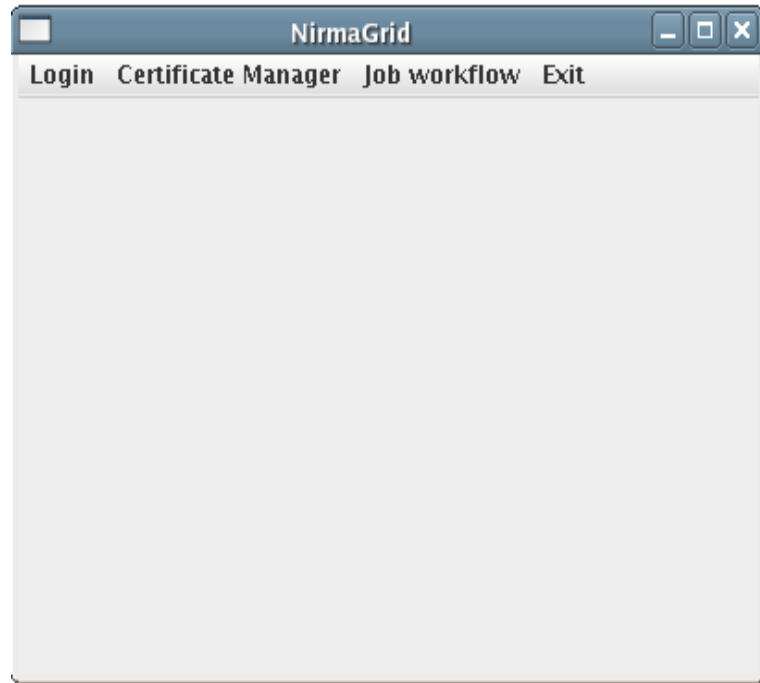


Figure 9.4: welcome

- Run and submit a job.
 - Suspend job which will modify a cluster queue.
 - This will create a checkpointing file which will save the state of a particular.
 - Status will show one job is pending shown in 9.8.
- V** CancelJob class Terminates a particular job with the help of Job ID and name of the job shown in figure 9.9.

9.5 Performance Measurement

Time without Agents	Time with Agents
0.0616	0.0475
3.397	3.20
0.0458	0.0408
0.383	0.375
2.856	2.756

Table I: Performance Comparison of System having Agents and without Agents

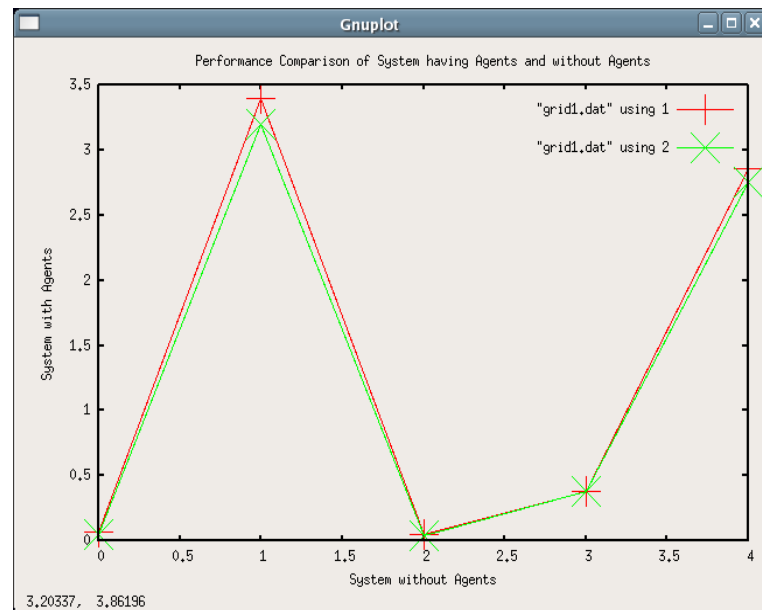


Figure 9.5: Performance Comparison of System having Agents and without Agents

Graph shows the comparison between job completion deployed in the grid environment with and without agents. It shows that the use of agents does not delay interactive job submission.

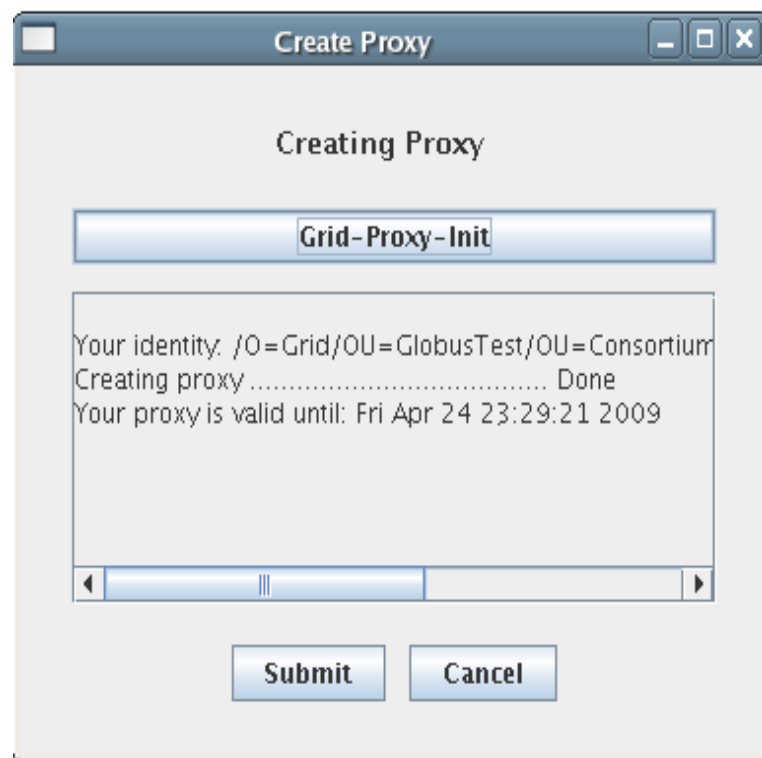
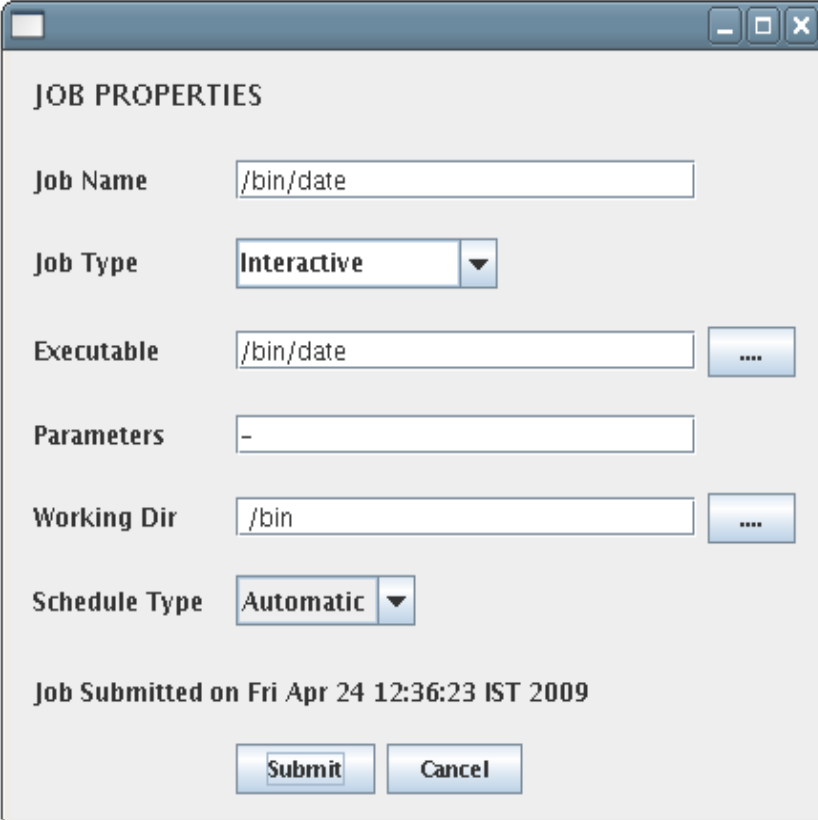


Figure 9.6: Create proxy



A screenshot of a 'JOB PROPERTIES' dialog box. The dialog has a title bar with standard window controls. It contains several fields: 'Job Name' with the value '/bin/date', 'Job Type' with a dropdown menu showing 'Interactive', 'Executable' with the value '/bin/date' and a browse button '...', 'Parameters' with the value '-', 'Working Dir' with the value '/bin' and a browse button '...', and 'Schedule Type' with a dropdown menu showing 'Automatic'. At the bottom, it displays 'Job Submitted on Fri Apr 24 12:36:23 IST 2009' and two buttons: 'Submit' and 'Cancel'.

JOB PROPERTIES

Job Name /bin/date

Job Type Interactive ▼

Executable /bin/date ...

Parameters -

Working Dir /bin ...

Schedule Type Automatic ▼

Job Submitted on Fri Apr 24 12:36:23 IST 2009

Submit Cancel

Figure 9.7: Job Submission

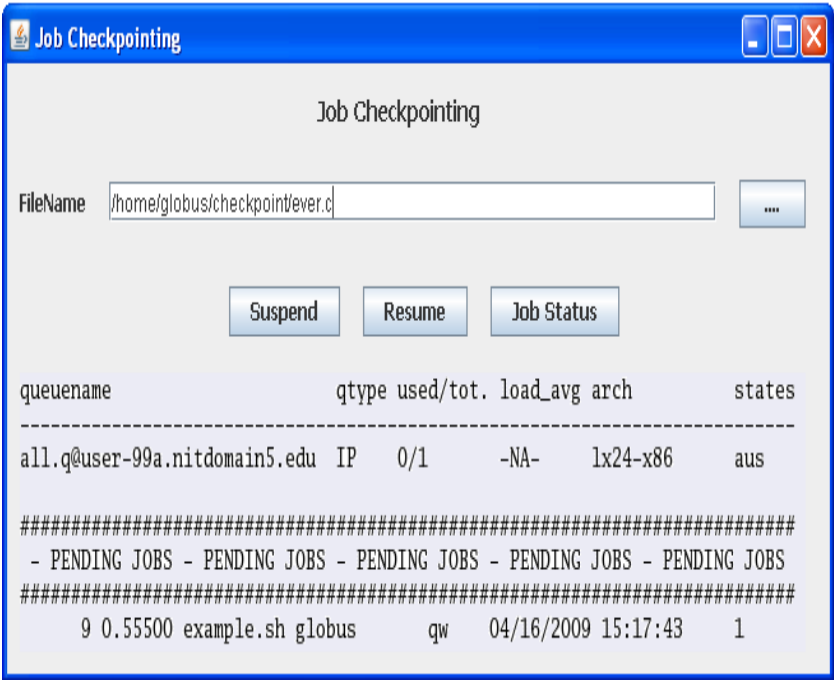


Figure 9.8: Job Checkpointing



Figure 9.9: Job Termination

Chapter 10

Conclusion and Future Scope

10.1 Conclusion

It is envisaged that the grid infrastructure will be a large-scale distributed system that will provide high-end computational and storage capabilities to differentiated users. In this project Globus Toolkit is used to establish the grid platform using desktop machines. Further more an agent based interactive job management system is developed to incorporate the concept of agent in the grid.

Developed system provides an interactive job management in grid environment. It works on the concept of the agent. A new job is assigned to an entity called Initiator agent. All the work carried out by that job is monitored by the Master agent through its various sub agents. This system provides the graphical user interface for submitting, monitoring, Suspending, Resuming and destroying various grid jobs. It also shows the current status of submitted and executing jobs.

Complete working is carried out by multiple agents. The object oriented methodology adopted provides platform independency for the jobs. It also increases the performance as compared to normal execution of the job as jobs are already assigned their respective agents and they will be taking care of their assigned jobs.

10.2 Future Scope

As far as this project is concerned, it can have many enhancements.

Globus Toolkit has itself got a set of services, which help us in doing several things like submission of jobs, getting resource attributes etc. We can ourselves design, implement and deploy a service which will do some particular work. We can have our own application as a service, and this service can be accessed by any node present within the grid.

Though it provides very good results while running large or big jobs which require very large computation resources, it is not advised to use the smaller jobs for execution as there will be a communication overhead if jobs are smaller. Thus high speed network or communication optimization can be considered as future work for this project.

Appendix A

GRID Environment Setup

A.1 Server Node

- To start PBS MOM `/opt/pbs/sbin/pbs_mom`
- PBS Server `/opt/pbs/sbin/pbs_server`
- PBS Scheduler `/opt/pbs/sbin/pbs_sched`
- `export GLOBUS_LOCATION=/opt/globus-4.0.1`
- `source $GLOBUS_LOCATION/etc/globus-user-env.sh`
- `export JAVA_HOME=/opt/jdk1.5.0_12`
- `export PATH=$JAVA_HOME/bin:$PATH`
- `export X509_USER_CERT=/opt/globus-4.0.1/etc/containercert.pem`
- `export X509_USER_KEY=/opt/globus-4.0.1/etc/containerkey.pem`

A.1.1 While Starting Container at server

Before Starting the container check if postgresql service is running properly or not.
Follow the given steps

- check if any other instance of postgres is running using **ps aux | grep postgres**.
If any other instance is running, change user to postgres with **su postgres**.
- **kill -SIGTERM "process id"**
- **/usr/bin/postmaster -i -D /opt/pgsql/data & /opt/pgsql/logfile 2&1 &**
- Change user to globus **su globus**
- Now set Globus location environment variables and java in PATH.
- **export GLOBUS_OPTIONS=-Xmx512M**
- **\$GLOBUS_LOCATION/bin/globus-start-container**

Now that we are sure the container is running correctly, use Ctrl+c to stop it. After a few seconds the container will stop. Start the container again but this time run it in the background.

```
opt/globus-4.0.1/bin/globus-start-container & $HOME/container.out 2&1 &
```

A.1.2 Starting globus-gridftp-server

- **export GRIDMAP=/opt/globus-4.0.1/etc/grid-mapfile**
- Start the server using the -p flag to run it on the default port of 2811 -
/opt/globus-4.0.1/sbin/globus-gridftp-server -p 2811 -S

A.1.3 Working with MDS

GT4 contains WebMDS, it doesn't have simple MDS. To work with MDS and get resource information TOMCAT must be installed on machine. To start TOMCAT SERVER:

- Change Directory to installed one.
- **sudo ./startup.sh**
- Same way to stop tomcat service type command `"./shutdown.sh"`

A.2 SGE Container node

- Set Globus location environment variables and java in PATH.
- `export SGE_ROOT=/opt/sge-root`
- start `./sgemaster` as root

A.2.1 Starting CONDOR

- `export CONDOR_CONFIG=(Condor_installation_dir)/etc/condor_config`
- `export PATH=$PATH:(Condor_installation_dir)/bin`
- `export PATH=$PATH:(Condor_installation_dir)/sbin`

To start CONDOR start `condor_master` on central manager nodeC by running `"condor_master"` command as globus user. Then do `"condor_on"` which will spawns all condor daemons.

Check to see if all the central manager daemons are running correctly with `ps aux | grep condor_`

A.3 Client node

As this is a client node no other environment variables are necessary. Make sure that GLOBUS_LOCATION and JAVA is in PATH.

Appendix B

TroubleShooting

Error: while GT4 make - **UNTAR FAILED**

Sol: It can be done by editing the file in `$GLOBUS_LOCATION/var/lib/perl/Grid/GPT/` where `$gunzip` is being set, it's called `LocalEnv.pm`. The root cause of the error is that `gunzip -version` was returning "gzip" in the version string instead of "gunzip".

Error: Starting Container gives error **Check if port number and host are correct and postmaster is accepting tcp/ip connections.**

Sol: Check if another instance of postgresql is not running. Check the log file for hint. If this is so change user to postgres use **kill -SIGTERM "process id"** and start postmaster again.

Error: cannot create regular file `"/opt/globus-4.0.1/etc/hostcert.pem"`: Permission denied (while signing certificate)

Sol: when `grid-cert-request` is run it creates three files `hostcert_request.pem`, `hostkey.pem`, `hostcert.pem`. Here `hostcert.pem` file is empty remove that file.

Error: while running command `globus-url-copy -vb gsiftp` gives **globus_xio:Unable to connect to nodeb.grid:2811**

Sol: start gridftp server using command “globus-gridftp-server -S -p 2811”

Error: “globusrun-ws -submit -streaming -F https://hostname:8443/wsrp/services/ManagedJobFactoryService -c /usr/bin/whoami” ERROR:Delegating user credentials...Failed. globusrun-ws: Error trying to delegate globus_xio: Unable to connect to hostname:8443 globus_xio: System error in connect: Connection refused globus_xio: A system call failed: Connection refused

Sol: copy globus-host-ssl.conf , globus-user-ssl.conf , grid-security.conf to /etc/grid-security/certificates and globus-user-ssl.conf.jhashnoç and globus-host-ssl.conf.jhashnoç to /etc/grid-security/.

Error: GRAM Job Submission failed because the connection to the server failed (check host and port) (error code 12)

Sol: Your client is unable to contact the gatekeeper specified. Possible causes include:

* The gatekeeper is not running * The host is not reachable. * The gatekeeper is on a non-standard port.

Error: globus-mds-start is not found in \$GLOBUS_LOCATION/sbin

Sol: GT4 uses web-mds other than MDS follow “webmdsAdminGuide” to start MDS service.

Error: Starting Condor daemons Error in MasterLog “can’t obtain lock on Instance-Lock”

Sol: The condor_master process tries to lock a file when it starts up to prevent you from starting multiple instances of the condor daemons. There are two instances of condor_master running so first stop condor_master and then it start again..

Error: Running condor_submit gives ERROR: submitting jobs as user/group 0 (root) is not allowed for security reasons.

Sol: When we say to run the daemons as root, that means you **start** them as root. They then switch to a non-root user whenever they don't need root power. This is to reduce the risk of screwing up the system if there's a bug or other problem. When the daemons need to do something that requires root power (like starting a job as the user), the daemons switch to root, do the deed, then return to non-root. When we say to run the daemons as root, that means you **start** them as root. They then switch to a non-root user whenever they don't need root power. This is to reduce the risk of screwing up the system if there's a bug or other problem. When the daemons need to do something that requires root power (like starting a job as the user), the daemons switch to root, do the deed, then return to non-root. Also during configuration you have to give ownership to globus user using command:

Condor_configure -owner=globus -type=submit,manager

This will allow globus user to submit job.

Appendix C

List of Websites

1 Globus Toolkit

<https://www.globus.org>

2 Globus toolkit installation guide

<https://www.globusconsortium.org/tutorial>

3 Globus Research Papers

<http://www.globus.org/alliance/publications/papers.php>

4 Globus Toolkit 4.0.1 Manual

<http://www.globus.org/toolkit/docs/development/4.0.1/>

5 Condor website

<http://www.cs.wisc.edu/condor>

6 Condor Manual

<http://www.cs.wisc.edu/condor/manual/v7.0/condor-V7.0.5-Manual.pdf>

7 PBS website

<http://www.openpbs.org>

References

- [1] D. Minoli, “A networking approach to grid computing,” Wiley Publications, 2006.
- [2] I. Foster, N. Jennings, and Kesselman, “Brain meets brawn: why grid and agents need each other,” 2004.
- [3] J. Cao, D. P. Spooner, J. D. Turner, S. A. Jarvis, “Agent-based resource management for grid computing,” IEEE/ACM International Symposium, 2002.
- [4] H. XIAO, H. WU, X. CHI, S. DENG, and H. ZHANG, “An implementation of interactive jobs submission for grid computing portals,” Supercomputing Center, Computer Network Information Center.
- [5] K. Jurasovic, G. Jezic, and M. Kusek, “A performance analysis of multi-agent systems,” 2006.
- [6] I. Foster and C. Kesselman in *The GRID 2: Blueprint for a New Computing Infrastructure*, November 2003.
- [7] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, and A. Neukoetter in *Introduction to Grid with Globus, IBM redbooks*.
- [8] R. Kumar, V. Talwar, and S. Basu, “A resource management framework for interactive grids,” Accepted in HPDC.
- [9] J. Chen, Y. Wu, M. Li, and B. Hui, “Multi-agent system-based hierarchy grid middleware,” University of California.
- [10] R. V. van Nieuwpoort, J. Maassen, R. Hofman, T. Kielmann, and H. E. Bal, “An efficient java-based grid programming environment,” November 2002.