QoS Based User Driven Scheduler For Grid

By

Sanjay Patel

08MCE013

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

QoS Based User Driven Scheduler For Grid

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Sanjay Patel

08MCE013

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2010

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Computer Science & Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Sanjay Patel

Certificate

This is to certify that the Major Project entitled **"QoS Based User Driven Scheduler For Grid"** submitted by **Sanjay Patel(08MCE013)**, 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 him 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,Senior Associate Professor,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. S.N. Pradhan
Professor,P.G.Coordinator,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

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

Abstract

As grids are in essence heterogeneous, dynamic, shared and distributed environments, managing these kinds of platforms efficiently is extremely complex. A promising scalable approach to deal with these intricacies is the design of self-managing of autonomic applications. Autonomic applications adapt their execution accordingly by considering knowledge about their own behavior and environmental conditions. QoS based User Driven scheduling for grid that provides the self-optimizing ability in autonomic applications. Computational grids to provide a user to solve large scale problem by spreading a single large computation across multiple machines of physical location.

QoS based User Driven scheduler for grid also provides reliability of the grid systems and increase the performance of the grid to reducing the execution time of job by applying scheduling policies defined by the user. The main aim of this project is to distribute the computational load among the available grid nodes and to developed a QoS based scheduling algorithm for grid and making grid more reliable. Grid computing system is different from conventional distributed computing systems by its focus on large-scale resource sharing, where processors and communication have significant influence on Grid computing reliability. Reliability capabilities initiated by end users from within applications they submit to the grid for execution.Reliability of infrastructure and management services that perform essential functions necessary for grid systems to operate, such as resource allocation and scheduling.

Acknowledgements

First of all, I would like to thank my principal supervisor **Prof.Madhuri Bhavsar** for her advice, encouragement and guidance throughout my Major Project. I am grateful to her for motivating and inspiring me to go deeply into the field of Grid computing and workflows and supporting me throughout the life cycle of Major Project. This research environment would not be possible without the determination of my supervisors who have helped create the best study conditions.

I like to give my special thanks to **Dr. S. N. Pradhan**, P.G. Coordinator, Department of Computer Science and Engineering,Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the project.

I am also thankful to **Dr.Ketan Kotecha**,Director,Institute of Technology for his kind support in all respect during my study.

I am also very thankful to other **staff members and to my friends** for their support. And at this moment, I would like to express my appreciation to my family members for their unlimited encouragement and support.

- Sanjay Patel

08MCE013

Contents

List of Figures

Abbreviation

QoS Quality of Service

GRAM Globus Resource Allocation Manager

MDS Metacomputing Directory Service

cMAs Cellular Memetic Algorithms

GSI Globus Security Infrastructure

OGSA Open Grid Service Architecture

AMS Application Management System

SDK Software Development Kit

FCFS First Come First Served

MRM Markov Reward Models

CAS Code Access Security

Chapter 1

Introduction

1.1 What is Grid?

A grid is a system that

- **Coordinates resources that are not subject to centralized control**
  Grid integrates and coordinates resources and users that exist within different control domains.

- **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.

- **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.

1.2 What is Grid Computing?

Grid computing is an interesting research area that integrates geographically-distributed computing resources into a single powerful system. Many applications can benefit from such integration . Examples are collaborative applications, remote visualization and the remote use of scientific instruments. Grid software supports such applications by addressing issues like resource allocation, fault tolerance, security, and heterogeneity. Parallel computing on geographically distributed resources, often called distributed supercomputing, is one important class of grid computing applications. Projects such as SETI@home, Intels Philanthropic Peer-to-Peer Program for curing cancer and companies such as Entropia show that distributed supercomputing is both useful and feasible.

Grid computing is a kind of parallel computing that enables the sharing, selection, and aggregation of geographically distributed autonomous resources, at runtime, as a function of availability, capability, performance, cost, and users quality-of-service requirements . One of the services that a Grid can provide is a computational service. Computational services execute jobs in a distributed manner . A Grid providing computational service is often called Computational Grid . We propose to implement the Grid using the visual studio Technology. Also we will implement the scheduling algorithm for grid. And then we will see the throughput of our grid for that particular application.

1.3 About Computational Grid

A computational Grid consists of a set of resources, such as computers, networks, online instruments, data servers or sensors that are tied together 12 by a set of common services which allow the users of the resources to view the collection as a seamless computing/information environment. The standard Grid services include

- Security services which support user authentication, authorization and privacy
- Information services, which allow users to see what resources (machines, software, other services) are available for use,
- Job submission services, which allow a user to submit a job to any compute resource that the user is authorized to use,
- Co-scheduling services, which allow multiple resources to be scheduled concurrently,
- User support services, which provide users access to "trouble ticket" systems that span the resources of an entire grid.

Grid services utilize the available computational resources so that tasks are run on whatever machine currently has available capacity. A grid also allows a single large computation to be spread across several machines, each of which is executing some portion of the computation. This can be done by:

- Breaking up the tasks into smaller tasks
- Assigning the smaller tasks to multiple hosts to work on simultaneously, coordinating with each other.

1.4 Computing Environment

The difference between standard computing and the grid can be seen from the following figure:

1.4.1 Standard Computing Environment

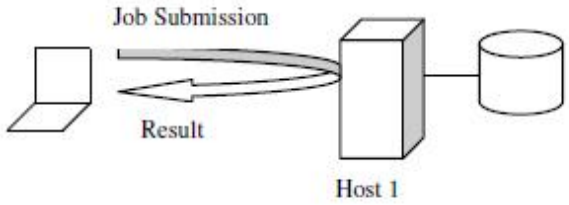

Figure 1.1: Standard Computing Environment

1.4.2 Grid Computing Environment

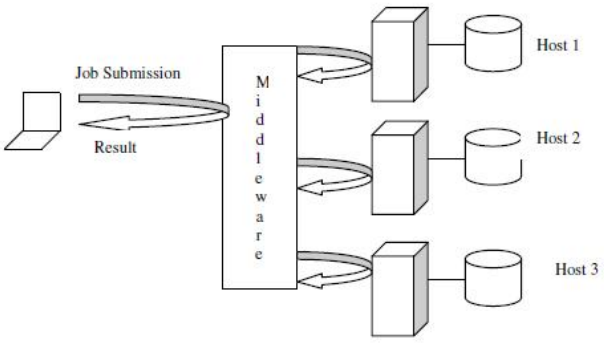

Figure 1.2: Grid Computing Environment

1.5 Why Grid?

In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle.

1.6 Problem Statement

As grids are in essence heterogeneous, dynamic, shared and distributed environments, managing these kinds of platforms effciently is extremely complex. A promising scalable approach to deal with these intricacies is the design of self-managing or autonomic applications. Autonomic applications adapt their execution accordingly by considering knowledge about their own behaviour and environmental conditions.QoS based User Driven scheduling for grid that provides the self-optimizing ability in autonomic applications.

1.7 Objective

The main objective of this project is to provide the self-optimizing and QoS driven scheduling ability in autonomic applications desired by the user and also developed a QoS based scheduling algorithm for grid and making grid more reliable.

1.8 Scope Of The Project

The User Driven Scheduler is intended to work as an resource managing module,queuing and scheduling of the Grid. The scheduler will offer managing batch jobs on Grid by scheduling CPU time according to user utility rather than system performance considerations.

Chapter 2

Literature Survey and Important observations

2.1 Grid Scheduling

In this work exploit the capabilities of Cellular Memetic Algorithms (cMAs) for obtaining efficient batch schedulers for Grid Systems. A careful design of the cMA methods and operators for the problem yielded to an efficient and robust implementation. Our experimental study, based on a known static benchmark for the problem, shows that this heuristic approach is able to deliver very high quality planning of jobs to Grid nodes and thus it can be used to design efficient dynamic schedulers for real Grid systems. Such dynamic schedulers can be obtained by running the cMAbased scheduler in batch mode for a very short time to schedule jobs arriving to the system since the last activation of the cMA scheduler [2].

Grid Services Scheduling is a challenging problem under Open Grid Service Architecture (OGSA). A graph theory formal description is introduced into the Service Grid Model in this paper. The necessary and sufficient condition of complete matching of user job and service resources has been given and proved. Optimal Solution to matchmaking of grid jobs and grid services is developed based on the running time

weight matrix, and the arithmetic has been verified by simulation analysis which proved to be more efficient than the alike arithmetic. The arithmetic has been implemented and running well. Along with the evolvement of computation technology and the Internet technology, Grid technology already became the core of new generation of network computation environment. Local Grid, also called Clusters, is placed at the center of the E-Commerce infrastructure with increasing requirements for providing service differentiation and performance assurance. To that end, Grid Services running on Clusters must have mechanism and policies for establishing and supporting QoS(Quality of Services) [3].

- No common and generic Grid scheduling system
- Used as a foundation for designing common Grid scheduling infrastructures

In the past years, many Grids have been implemented and became commodity systems in production environments. While several Grid scheduling systems have already been implemented, they still provide only ad hoc and domain-specific solutions to the problem of scheduling resources in a Grid. However, no common and generic Grid scheduling system has emerged yet. In this work we identify generic features of three common Grid scheduling scenarios, and we introduce a single entity that we call scheduling instance that can be used as a building block for the scheduling solutions presented [4].

Providing Self optimizing ability for autonomic applications.A promising scalable approach to deal with these intricacies is the design of self-managing or autonomic applications. Autonomic applications adapt their execution accordingly by considering knowledge about their own behaviour and environmental conditions. This paper focuses on the dynamic scheduling that provides the self-optimizing ability in autonomic applications. Being distributed, collaborative and pro-active, the proposed hierarchical scheduling infrastructure addresses important issues to enable an efficient execution in a computational grid. Unlike other approaches, the cooperative, hybrid

and application-specific strategy deals effectively with task dependencies. Several experiments have been analyzed in real grid environments highlighting the efficiency and scalability of the proposed infrastructure. This paper presents an intra-site dynamic scheduling heuristic for tightly coupled parallel applications represented by DAGs [5].

Present a scheduling algorithm which showed performance improvements to the original algorithm shipped with Alchemi grid software. Computational grids are useful tools for bringing super computing power to users by using idle resources in the network. In the following paper we give a short overview of architecture of the Alchemi grid developed on .Net platform. We created a grid application, which utilizes Rabin Karp string searching algorithm to test Alchemi grid performances in situation when requests put diverse demands for computing resources to the different grid nodes. Scheduling and dispatching jobs to the computing resources is a critical activity of the grid software. We present a scheduling algorithm which showed performance improvements to the original algorithm shipped with Alchemi grid software [1].

The main objective of these work is to schedule Enterprise Grid workloads so as to minimise the costs, while ensuring desired Quality-of-Service with a certain degree of confidence. Grids provide infrastructure for intensive computations and storage of shared large scale database across a distributed environment. Although grid technologies enable the sharing and utilization of widespread resources, the performance of parallel applications on the Grid is sensitive to the effectiveness of the scheduling algorithms used. Scheduling is the decision process by which application components are assigned to available resources to optimize various performance metrics. In this paper we discuss how economy and quality of service plays important role in scheduling resources. Various Grid scheduling algorithms are discussed from different points of view, such as static vs. dynamic policies, objective functions, Quality of service constraints, strategies dealing with dynamic behavior of resources and so on. After discussing existing algorithms a new Failure rate-cost and time optimization algorithm is been proposed which will guarantee quality of service [6].

2.2 Algorithm and Methods

2.2.1 Cellular Memetic Algorithm

Description:

- In Memetic Algorithms (MAs) the population of individuals could be unstructured or structured.

- As in the case of other evolutionary algorithms, cMAs are high level algorithms whose description is independent of the problem being solved. As it can be seen, this template is quite different from the canonical cGA approximation , in which individuals are updated in a given order by applying the recombination operator to the two parents and the mutation operator to the obtained offspring. In the case of the algorithm proposed in this work, mutation and recombination operators are applied to individuals independently of each other, and in different orders. This model was adopted after a previous experimentation, in which it performed better than the cMA following the canonical model for the studied problems. After each recombination (or mutation), a local search step is applied to the newly obtained solution, which is then evaluated. If this new solution is better than the current one, it replaces the latter in the population. This process is repeated until a termination condition is met [10].

cMA methods and operators: The performance of any cMA heavily depends on the design and implementation of the methods and operators.

- Solution representation

- Fitness

- Population initialization

2.2.2 Graph Theory

Description:

- Services scheduling mainly acts according to some high-level application QoS parameters to carry on, for instance, the complete time, the reliability or the service cost and so on
- To develops a QoS-aware Grid Services Scheduling optimal algorithm based on the complete time weight matrix.

The services scheduling process does not care about the core function itself of resources and services, but care about the way of the core function execution, for instance the start time of requested operation, the running time needed to complete, its cost or expense and so on. Therefore, we establish a scheduling service and a sensor service independent of the resources provider and the resources consumer, thus forms a simplified four unit Grid Services scheduling model. This model including following componets [3].

- **Job Manager:** The agent for the service consumer in the service grid . It plays the role of entrance of the end users, provides the interfaces to job submitting, job inquiring and job deletion operation.
- **Service Manager:** The agent for the service provider in the service grid . It manages the resources, and is responsible for the dispatchment appropriate job running on the resources .
- **Scheduler:** Its function is to seek the suitable match between the services consumer and the services provider through some certain of algorithms, and to inform the both sides matched to carry on the binding operation.
- **Sensor:** Its function is to monitor the running state of resources and jobs in real time. Monitor making is the process to log the state data of the resources and jobs. As a independent service,sensor can monitor the state , such as the

starting of job, the running of job, the suspending of job, the activation of job, the end of job .

2.2.3 Scheduling Instance

Description:

- A scheduling instance is defined as a software entity that exhibits a standardized behavior with respect to the interactions with other software entities
- The scheduling instance is the basic building block of a scalable, modular architecture for scheduling tasks, jobs,workflows, or applications in Grids.

In this context, a scheduling instance is defined as a software entity that exhibits a standardized behavior with respect to the interactions with other software entities (which may be part of a GSA implementation or external services). Such scheduling entities cooperate to provide, if possible, a solution to scheduling problems submitted by users, e.g. the selection, planning and reservation of resource allocations for a job . The scheduling instance is the basic building block of a scalable, modular architecture for scheduling tasks, jobs, workflows, or applications in Grids. Its main function is to find a solution to a scheduling problem that it receives via a generic input interface. To do so, the scheduling instance needs to interact with local resource management systems that typically control the access to the resources. If a scheduling instance can find a solution for a submitted scheduling problem, the generated schedule is returned via a generic output interface. From the examples depicted above it is possible to derive a high level model of operations for a generic set of cooperating scheduling instances. To provide a solution to a scheduling problem, a scheduling instance can exploit several options [4].
From a component point of view abilities as described above are expressed as interfaces. In general, the interfaces of a scheduling instance can be divided in two main

categories: functional interfaces and non-functional interfaces. The former are necessary to enable the main behaviors of the scheduling instance, while the latter are concerned with the management of the instance itself (creation, destruction, status notification, etc.). We want to highlight that we consider only the functionalities that must be directly exploited to support a general scheduling architecture; for example, security services are from a functional point of view not strictly needed to schedule a job, so they are considered external services or non-functional interfaces.

2.2.4 Easy Grid AMS (Application Management System)

Description: This section briefly describes the EasyGrid AMS that is used in this work to manage the execution of a parallel MPI application on the computational grid. The EasyGrid AMS implements dynamic process creation and is automatically embedded into the MPI parallel application. It is not de- pendent on other grid system middleware, requiring only the Globus Toolkit and the LAM/MPI library to be installed.

The EasyGrid AMS employs a distributed hierarchy of management processes each composed of four layers with specific functions: process management, application moni- toring, dynamic scheduling and fault tolerance. Each MPI application has its own three level hierarchical management system composed of: a single Global Manager (GM), at the top level, which supervises the sites in the grid where the application is running; at each of these sites, a Site Manager (SM) is responsible for the allocation of the application pro- cesses to the resources available at that site; and finally, the Host Manager (HM), one for each resource, takes on the responsibility for creation and execution of the MPI appli- cation processes. The management processes have minimal intrusion because they behave like event-driven daemons, only consuming CPU cycles to process messages [5].

- The EasyGrid AMS implements dynamic process creation and is automatically embedded into the MPI parallel application.

- The EasyGrid AMS employs a distributed hierarchy of management processes each composed of four layers with specific functions: process management, application monitoring, dynamic scheduling and fault tolerance.

2.2.5 Dedicated Nodes Algorithms

- **Rolling Hash Algorithm:** It extracts all N word phrases from the text1 and stores them into PhraseList collection.As described in [12], first a hash of substring1 that must be matched is calculated. Then, a substrin2 is selected where Length(Substring1) = Length(Substring2). Hash value of susbstring2 is compared to the calculated hash value of susbtring1. Comparing hash values gives a speed improvement to a direct string comparison because it is not necessary to loop through substring1 and substring2 to compare a character by character. The disadvantage is that the same hash value can represent two different strings. This situation is called a collision and it has to be resolved. With an efficient hash algorithm it is possible to achieve O(n) time, as described in Rolling hash algorithm [1].

- **Rabi Karp Algorithm:** The algorithm uses a rolling hash to compare a string1 of the length N with all possible substrings of the length equal to M in the given text. In order to compare Text1 and Text2 for all phrases with N words, the following algorithm is applied to Text1 to find all N word phrases. Phrases are grouped together by their character length to utilize rolling hash algorithm. It extracts all N word phrases from the text1 and stores them into PhraseList collection. This is an array of arrays of phrases with the same character length. The next step is to check if each of the phrases in the PhraseList exists in the Text2. If we have M phrase arrays with each having a varying number of phrases with the same character length, each phrase array from the Text1 can be checked efficiently for a presence in Text2 with the described Rabin-Karp algorithm. Obviously, we can utilize the Alchemi grid to, instead of looping

sequentially M times, create M GThreads and submit them to the manager. As described in the introduction section, a mechanism for submitting tasks to the manager uses an instance of GApplication class [1].

- **Default Scheduling Algorithm:** The algorithm assigns each task to an available processing node. The first available executor is fetched from the executor storage. The first waiting thread with the highest priority is fetched from the thread storage. Dedicated schedule is created by pairing the executor and the thread. This information is used by a dispatcher to dispatch a job to the assigned executor. This type of scheduling is known in the literature as opportunistic matching. The algorithm assigns each task to an available processing node. The expected task execution time is not taken in the account. Execution time decreases as performances of the computing node increases. Execution time for a same task is a function of factors such as bandwidth between manager and executors and capabilities of the individual executors. According to good scheduling schemes should be able to exploit the differences in the bandwidth between manager and every individual executor and differences between performances of the executor nodes [1].

2.3 Conclusion

cMAs are a good choice for scheduling jobs in Computational Grids given that they are able to deliver high quality plannings in a very short time. This paper gives a four unit Grid Services scheduling model based on graph theory, develops a QoS-aware Grid Services Scheduling optimal algorithm based on the complete time weight matrix. The necessary and sufficient condition of complete matching of the grid jobs and grid services is given also. This algorithm has been used in the multimedia data transmission services in the electronic commerce to guarantee the QoS of reserved resources and running well. In this paper discuss the general architecture for grid scheduling. Several scheduling instance implementations can be composed to build

existing scheduling scenarios as well as new ones. This work presents a novel dynamic scheduling strategy that deals efficiently with the precedence relations that exist in tightly-coupled parallel applications.

To improve performance of Alchemi grid using different algorithm. The paper presents a simple heuristic scheduling algorithm that has a potential to improve performances in the Alchemi P2P desktop grid.In this paper, proposed a new Grid scheduling algorithm that minimizes the cost of execution of workflows, while ensuring that their associated Quality-of-Service constraints are satisfied.

Chapter 3

Existing Methodologies

A grid is created by installing Executors on each machine that is to be part of the grid and linking them to a central Manager component. The Windows installer setup that comes with the Alchemi distribution and minimal configuration makes it very easy to set up a grid.

3.1 Layered Architecture of Grid

Users can develop, execute and monitor grid applications using the .NET API and tools which are part of the Alchemi SDK. Alchemi offers a powerful grid thread programming model which makes it very easy to develop grid applications and a grid job model for grid-enabling legacy or non-.NET applications [1].

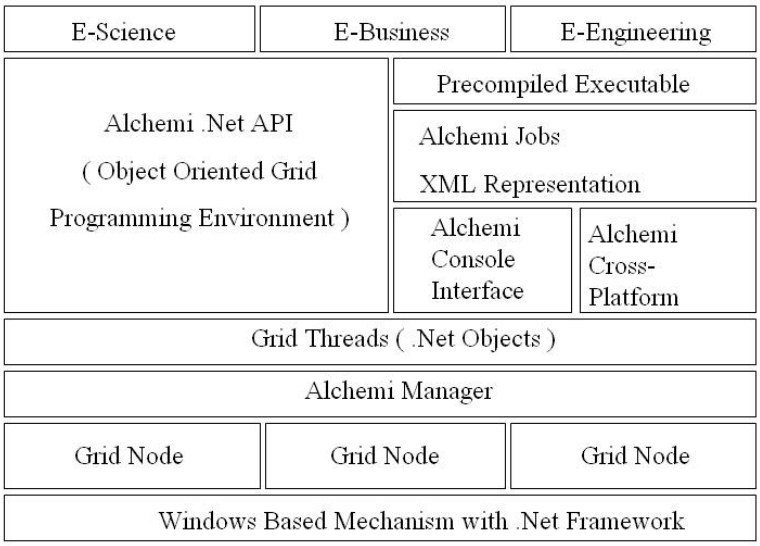

Figure 3.1: Layered Architecture of Grid

3.2 Alchemi Desktop Grid Framework

Alchemi layered architecture for a desktop grid computing environment is shown in Figure. Alchemi follows the master-worker parallel computing paradigm in which a central component dispatches independent units of parallel execution to workers and manages them. In Alchemi, this unit of parallel execution is termed grid thread and contains the instructions to be executed on a grid node, while the central component is termed Manager [1].

A grid application consists of a number of related grid threads. Grid applications and grid threads are exposed to the application developer as .NET classes / objects via the Alchemi .NET API. When an application written using this API is executed, grid thread objects are submitted to the Alchemi Manager for execution by the grid. Alternatively, file-based jobs (with related jobs comprising a task) can be created using an XML representation to grid-enable legacy applications for which precompiled executables exist. Jobs can be submitted via Alchemi Console Interface or Cross-Platform Manager web service interface, which in turn convert them into the grid

threads before submitting then to the Manager for execution by the grid.

3.2.1 Application Models

Alchemi supports functional and well as data parallelism. Both are supported by each of the two models for parallel application composition grid thread model and grid job model.

a. **Grid Thread Model:** Minimizing the entry barrier to writing applications for a grid environment is one of Alchemis key goals. This goal is served by an object-oriented programming environment via the Alchemi .NET API which can be used to write grid applications in any .NET-supported language.

   The atomic unit of independent parallel execution is a grid thread with many grid threads comprising a grid application (hereafter, applications and threads can be taken to mean grid applications and grid threads respectively, unless stated otherwise). The two central classes in the Alchemi .NET API are GThread and GApplication, representing a grid thread and grid application respectively. There are essentially two parts to an Alchemi grid application. Each is centered on one of these classes:

   - **Remote code:** code to be executed remotely i.e. on the grid (a grid thread and its dependencies)
   - **Local code:** code to be executed locally (code responsible for creating and executing grid threads).

   A concrete grid thread is implemented by writing a class that derives from GThread, overriding the void Start() method, and marking the class with the Serializable attribute. Code to be executed remotely is defined in the implementation of the overridden void Start() method.

   The application itself (local code) creates instances of the custom grid thread, executes them on the grid and consumes each threads results. It makes use of

an instance of the GApplication class which represents a grid application. The modules (.EXE or .DLL files) containing the implementation of this GThread-derived class and any other dependency types that not part of the .NET Framework must be included in the Manifest of the GApplication instance. Instances of the GThread-derived class are asynchronously executed on the grid by adding them to the grid application. Upon completion of each thread, a thread finish event is fired and a method subscribing to this event can consume the threads results. Other events such as application finish and thread failed can also be subscribed to. Thus, the programmatic abstraction of the grid in this manner described allows the application developer to concentrate on the application itself without worrying about "plumbing" details.

b. **Grid Job Model:**Traditional grid implementations have offered a high-level, abstraction of the "virtual machine", where the smallest unit of parallel execution is a process. In this model, a work unit is typically described by specifying a command, input files and output files. In Alchemi, such a work unit is termed job with many jobs constituting a task.

   Although writing software for the grid job model involves dealing with files, an approach that can be complicated and inflexible, Alchemis architecture supports it for the following reasons:

   - grid-enabling existing applications
   - Inter operability with grid middleware that can leverage Alchemi via the Cross Platform Manager web service

   Before submitting the task to the Manager, references to the embedded files are resolved and the files themselves are embedded into the task XML file as Base64-encoded text data. When finished jobs are retrieved from the Manager, the Base64-encoded contents of the embedded files are decoded and written to disk.

3.3 Existing Scheduling Mechanism

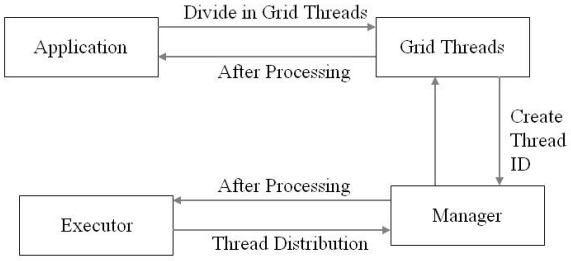

Figure 3.2: Existing Scheduling Mechanism

In existing scheduling mechanism all threads and computation done by system at kernel level. First of all application get divided in to different grid enabled process part and each small part called thread. When application divide in to different thread then system assign to each thread different thread id. Each thread id(thread) divide for computation to available grid nodes. After computation each grid nodes return back thread or computation results to the head node [1].

Chapter 4

Tools and Technique

4.1 Front End Tools

- Windows
- .Net Framework 3.0
- Microsoft Visual Studio C#.net
- Alchemi 1.0.6
- Perfwiz

4.2 Back End Tools

- Microsoft Sql Server 2005

4.3 Alchemi Toolkit Overview

There are four types of distributed components (nodes) involved in the construction of Alchemi grids and execution of grid applications Manager, Executor, User & Cross-Platform Manager [1].

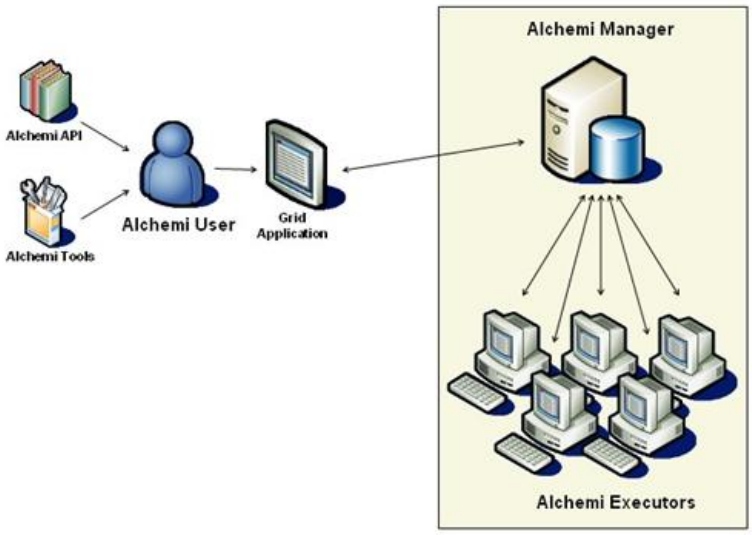

Figure 4.1: An Alchemi Grid

A grid is created by installing Executors on each machine that is to be part of the grid and linking them to a central Manager component. The Windows installer setup that comes with the Alchemi distribution and minimal configuration makes it very easy to set up a grid. An Executor can be configured to be dedicated (meaning the Manager initiates thread execution directly) or non-dedicated (meaning that thread execution is initiated by the Executor.) Non-dedicated Executors can work through firewalls and NAT servers since there is only one-way communication between the Executor and Manager. Dedicated Executors are more suited to an intranet environment and non-dedicated Executors are more suited to the Internet environment.

4.3.1 Distributed Components

Four types of nodes (or hosts) take part in desktop grid construction and application execution (see Figure). An Alchemi desktop grid is constructed by deploying a Manager node and deploying one or more Executor nodes configured to connect to the Manager. One or more Users can execute their applications on the cluster by connecting to the Manager. An optional component, the Cross Platform Manager

provides a web service interface to custom grid middleware. The operation of the Manager, Executor, User and Cross Platform Manager nodes is described below.

a. **Manager:**The Manager provides services associated with managing execution of grid applications and their constituent threads. Executors register themselves with the Manager, which in turn monitors their status. Threads received from the User 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 or submitted. 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 collected by the respective users. A scheduling API is provided that allows custom schedulers to be written.

   The Manager employs a role-based security model for authentication and authorization of secure activities. A list of permissions representing activities that need to be secured is maintained within the Manager. A list of groups (roles) is also maintained, each containing a set of permissions. For any activity that needs to be authorized, the user or program must supply credentials in a form of a user name and password and the Manager only authorizes the activity if the user belongs to a group that contains the particular permission.

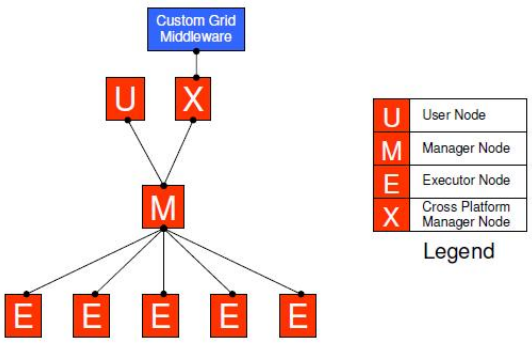

Figure 4.2: Distributed components and their relationships

As discussed previously, failure management plays a key role in the effectiveness

of a desktop grid. Executors are constantly monitored and threads running on disconnected Executors are rescheduled. Additionally, all data is immediately persisted to disk so that in the event of a crash, the Manager can be restarted into the pre-crash state.

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 explicitly by the user. For non-dedicated execution, there is one-way communication between the Executor and the Manager. In this case, the resource that the Executor resides on is managed on a volunteer basis since it requests threads to execute from the Manager. When two-way communication is possible and dedicated execution is desired the Executor exposes an interface so that the Manager may communicate with it directly. In this case, the Manager explicitly instructs the Executor to execute threads, resulting in centralized management of the resource where the Executor resides. Thus, Alchemis execution model provides the dual benefit of:

   - flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution
   - flexible deployment under network constraints i.e. the component can be deployment as nondedicated where two-way communication is not desired or not possible (e.g. when it is behind a firewall or NAT/proxy server).

   Thus, dedicated execution is more suitable where the Manager and Executor are on the same Local Area Network while non-dedicated execution is more appropriate when the Manager and Executor are to be connected over the Internet.

Threads are executed in a sandbox environment defined by the user. The CAS (Code Access Security) feature of .NET are used to execute all threads with the Alchemi GridThread permission set which can be specified to a fine-grained level by the user as part of the .NET Local Security Policy. All grid threads run in the background with the lowest priority. Thus any user programs are unaffected since they have higher priority access to the CPU over grid threads.

c. **User:**Grid applications are executed on the User node. The API abstracts the implementation of the grid from the user and is responsible for performing a variety of services on the users behalf such as submitting an application and its constituent threads for execution, notifying the user of finished threads and providing results and notifying the user of failed threads along with error details.

d. **Cross-Platform Manager:**The Cross-Platform Manager is a web services interface that exposes a portion of the functionality of the Manager in order to enable Alchemi to manage the execution of 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. In addition to support for the grid-enabling of legacy applications, the Cross-Platform Manager allows custom grid middleware to inter operate with and leverage Alchemi on any platform that supports web services.

Chapter 5

The Proposed Algorithm and Architecture

5.1 Process Sequence

There are four main entities in architecture, which are users, manager, schedulers and executor. A client is a user who submits a job to the system. A job refers to a collection of computation that the client wants to execute. The job is submitted by the client to the manager 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.

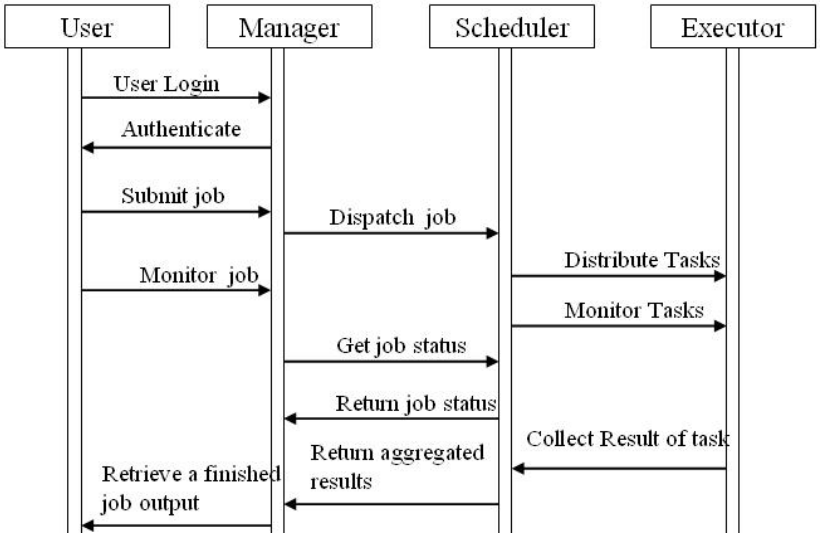

Figure 5.1: Job Processing Sequence

5.2 Design

A typical Grid consists of a number of services and a number of physical resources, including compute resources that are capable of hosting these services as well as storage resources, network resources etc. Grid applications are typically defined in terms of workflows, consisting of one or more tasks that may communicate and cooperate to achieve their objective. The job of the scheduler is to select a set of resources on which to schedule the tasks of an application, assign application tasks to compute resources, coordinate the execution of the tasks on the compute resources and manage the data distributions and communication between the tasks.

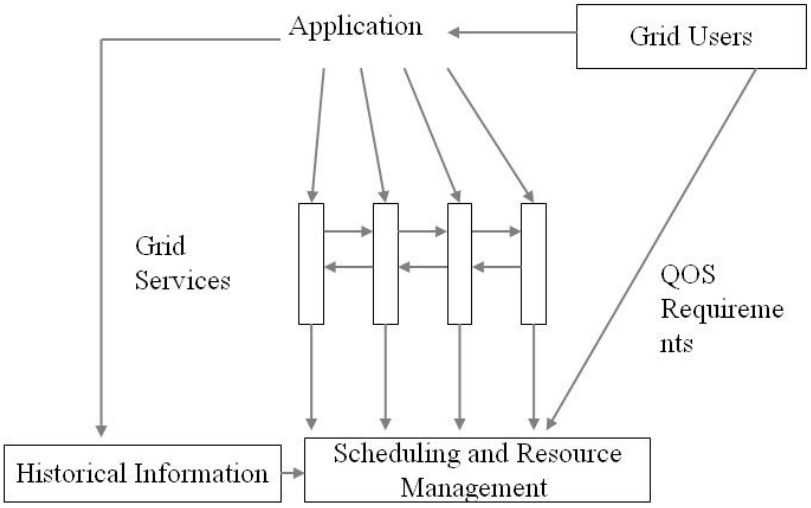

Figure 5.2: New Scheduling Mechamism

5.3 Algorithm Steps

a. List Out All the Resources

b. Sort all the resources with their success rate

c. Find resource with highest success rate

d. Assign job to that resource

e. If Success rate is same of two resources then compare both resources cosidering their time

f. Assign job to that resource whose time is minimum

g. Repeat steps 3 to 5 till all jobs have been assign resources

5.4 Algorithm Description

Failure of a resource while doing scheduling is not being considered at the time of allocating resources by the broker. Here, at the time of scheduling jobs, broker will consider only minimum cost of a resource along with MIPS of that resource. While doing scheduling, if resource fails to execute any job then such thing cannot be ignored when next time a job needs to be executed on that resource.So for a resource a new parameter is added as failure rate which will consider success rate of a resource. If a resource is having 100% failure rate then that means that whenever a job is scheduled on that resource then it will surely fail to execute that job on that resource. By default the value for failure rate should be zero. A history is maintained by the broker, which will keep track of this failure rate. If any failure happens then broker will update history of that resource, which will be considered at the time of next scheduling of jobs.

5.5 Flow Chart

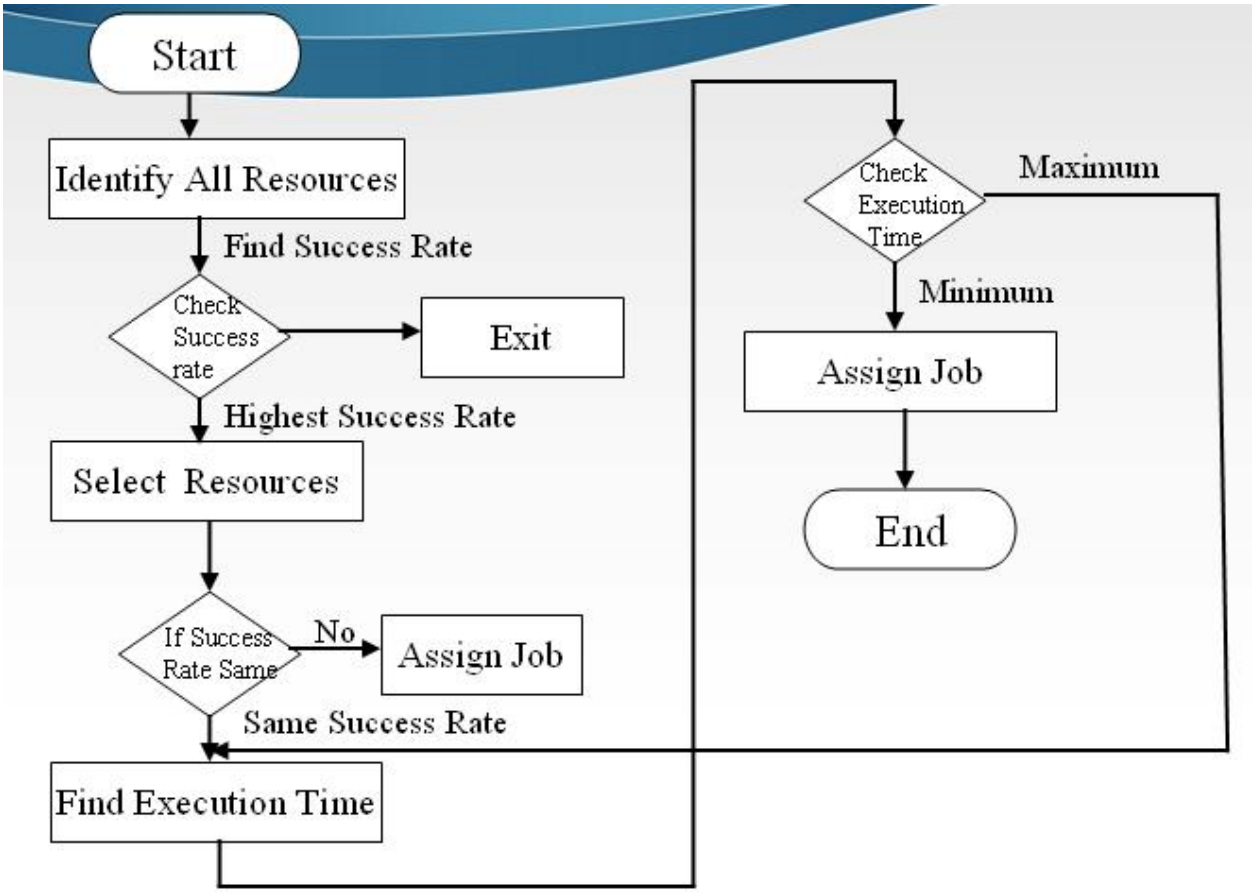

Figure 5.3: Flow Chart

In following figure given steps to identify best resource. First of all we identify all the resources in the grid. Then find success rate of each grid node.Compare success rate of each grid node and which grid node success rate is high then we select this resource for computation and also arrange grid node by success rate. but two grid node success rate is same then we also find execution time of each grid node. after find execution time of each grid node we compare execution time of grid node. we find minimum execution time of node that grid node we assign for computation. we rotate this steps.

5.6 Algorithm

Input: M = (J1, J2, ..., Jn), where
s1 $\geq$ s2 $\geq$ ... $\geq$ sn for LJF; //Largest Job First
t1 $\geq$ t2 $\geq$ ... $\geq$ tn for LTF; //Longest Time First
Output: θ =(θ1, θ2, ..., θn).
J=j1+j2+j3+.....+jn //Divide job in sub jobs
Initially,
(P1, P2 , ..., Pn) = (P1, P2, ..., Pn);
P = P1 + P2 + ... + P n . //Total no. of Processor
At time t = 0, or whenever a job is completed at time t
reclaim the Resources used by the job just
completed;
for (every Jd in M)**do**
if (si > P)
stop;
or skip Ji;
else
call a Resource allocation algorithm to get Resource
allocation ((Pj1, si,1), (Pj2, si,2), ..., (Pjn, si,n)); //Allocate Processor
generate the schedule of Ji as
θi = (t, (Pj1, si,1), (Pj2, si,2), ..., (Pjn, si,n));
P = P - si;
remove Ji from L;

Figure 5.4: Scheduling Algorithm

Chapter 6

Implementation

6.1 Grid Manager GUI

6.1.1 Display Grid Nodes

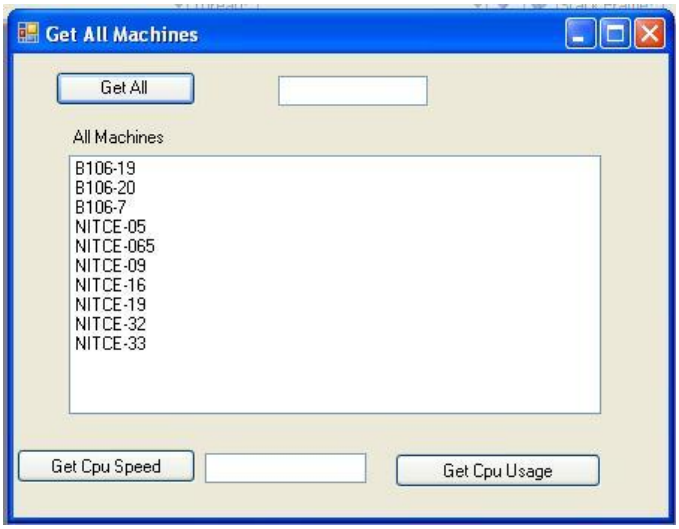

Figure 6.1: Display Grid Nodes

In this window i am trying to display different available grid nodes which is using in the computation.

6.1.2 Cpu Speed

Figure 6.2: Cpu Speed

In this window i am trying to display cpu speed of the grid nodes.

6.1.3 Cpu Usage

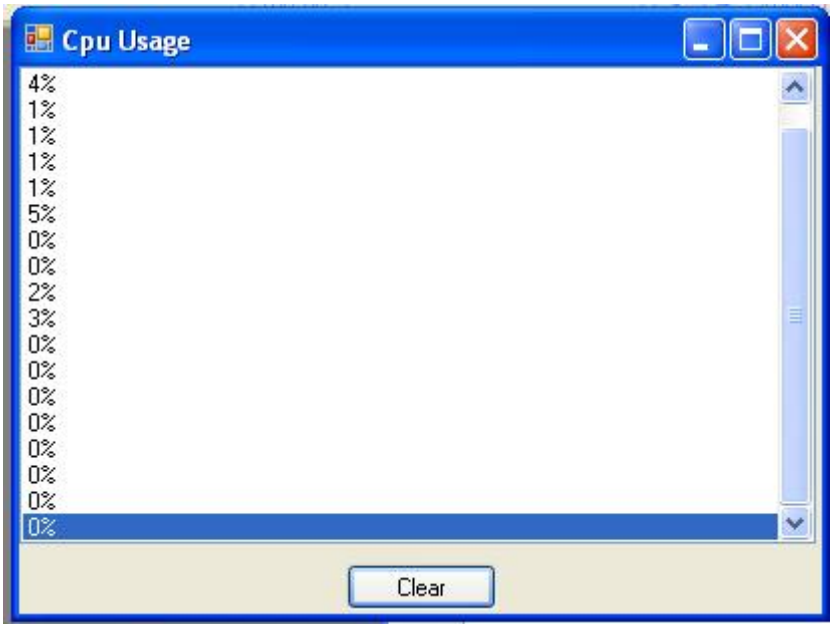

Figure 6.3: Cpu Usage

In this screen shows the cpu usage of the grid nodes.

6.2 Grid User GUI

- **Grid User Login**

Figure 6.4: Grid User Login

This window is used for the Grid user,using this window Grid user can login in the Grid system.

- **Grid User Main Window**

Figure 6.5: Grid User Main Window

This is the Grid user main window.Using this window Grid user can submit job to the manager for the computation. Another functionality of this window Grid User can View the Result of Submit job.

- **Submit Job**

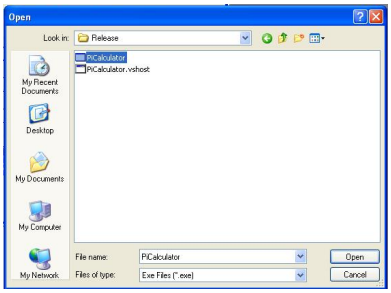

Figure 6.6: Grid User Submit Job

This window display the user submit the job which required for computation and also display the job path.

- **View Result**

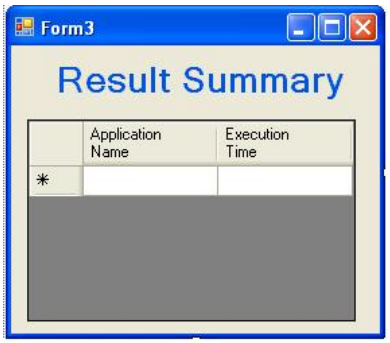

Figure 6.7: Grid User View Result

This display the no of application and their execution time for computation.

6.3 Users QoS Requirements

- **Time:**

  Minimize execution time to increase the performance.

- **Reliability:**

  No. of failures for execution of workflows.

- **Fidebility:**

  Measurement related to the quality of the output of execution

6.4 Implement Reliability in grid

- Scientific applications have diverse performance and reliability requirements that are often difficult to satisfy, given the variability of underlying resources. Availability can vary due to failure of one or more critical services, load on one or more resource components, recovery from a failure, etc. Moreover, as Grid and web services continue to evolve, rapidly changing software stacks with concomitant configuration and service reliability challenges exacerbate application execution times and failures.

- In addition higher levels of the software stack need the ability to clearly express performance and reliability expectations. Although there are tools and well understood mechanisms to monitor performance and ensure reliability, few tools allow users to express reliability policies from the applications perspective, map these to resource capabilities and cost models and enforce appropriate workflow scheduling, fault tolerance and recovery strategies [13].

6.4.1 Reliability Programming Models

a. **Master Worker :**In the master-worker paradigm the master decomposes the problem into small tasks and distributes these tasks for execution.

b. **Divide and Conquer :**The divide and conquer strategy partitions the problem into two or more smaller problems that can be solved independently and combined [14].

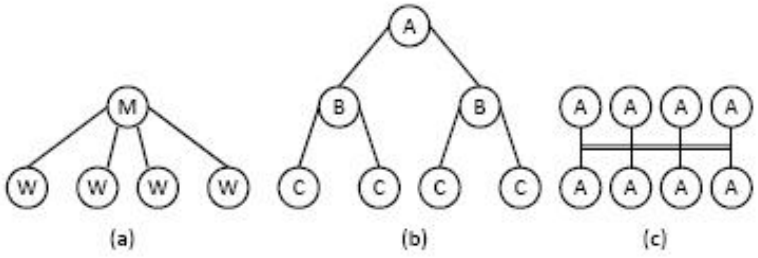

Figure 6.8: Three common programming models (a) Master Worker (b) Divide and Conquer (c) SPMD

c. **SPMD :**In the SPMD model, each task executes common code on different data. Failure of one task adversely affects the entire application, requiring global coordination.

6.4.2 Reliability Specification

In this section, we discuss the extensions required to the virtual grid description language to support reliability specifications. We define a high-level qualitative reliability metric space that can be used to request resources. The qualitative levels are mapped to well-defined quantitative reliability levels in the virtual grid to enable runtime monitoring and adaptation.

Define a 5-point qualitative reliability scale that maps to quantitative levels of availability as follows: [11]

- **High Reliability (90-100%)**
- **Good Reliability (80-89%)**
- **Medium Reliability (70-79%)**
- **Low Reliability (60-69%)**
- **Poor Reliability (59-0%)**

6.4.3 Performability Analysis

- Grid systems are often able to survive the failure of one or more components and continue to provide service, but with reduced performance. Such behavior and status of systems with multiple interacting components is typically captured using stochastic process modeling [12].

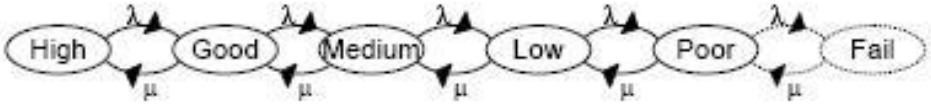

Figure 6.9: Markov chain for the resource performance and reliability states

- The probability of staying in a certain state with respect to transition rates between states is used to quantify system performance and reliability. Markov Reward Models (MRM) are typically used to model gracefully degradable systems and capture joint performance and system reliability. A Markov reward model consists of a Markov chain that describes a systems possible states and an associated reward function.

6.5 Divide and Conquer

- The divide and conquer strategy partitions the problem into two or more smaller problems that can be solved independently and combined. Each subtask may be further split into separate tasks. Unlike the master-worker model, the subtasks are interdependent.

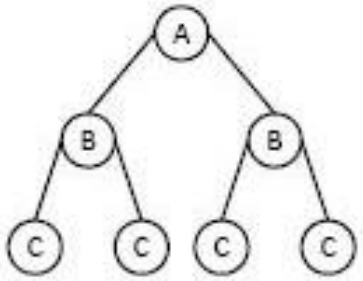

Figure 6.10: Divide and Conquer

- Hence the performance and reliability requirements (e.g. for the communication links) might vary significantly from the master-worker model.

6.6 Divide and Conquer Algorithm

- **Description :**Recursively partition a problem into subprogram of roughly equal size. If subprogram can be solved independently, there is a possibility of significant speed up by parallel computing.

- **Functional Structure of Divide and Conquer Algorithm:**The divide and conquer strategy partitions the problem into two or more smaller problems that can be solved independently and combined.

6.7 Merge Sort

- **Merge Sort :**Given two ascending sorted sublists, L[0:N-1] and L[N:2N-1]
- **Recursive Merge Sort :**Given a list, divide it in half, sort the two halves (recursively),and then merge the two list together.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using Alchemi.Core;
using Alchemi.Core.Manager.Storage;
using log4net;
namespace DivideMerge
{
   public class Merge : Form
   {
      int i,j,k,t,middle,temp[N];
if (left < right) {
middle = (left + right)/2;
mergesort(list,left,middle);
mergesort(list,middle+1,right);
k = i = left; j = middle+1;
while (i<=middle && j<=right)
temp[k++] = list[i]<list[j] ? list[i++] : list[j++];
t = i>middle ? j : i;
while (k <= right) temp[k++] = list[t++];
for (k=left; k<=right; k++) list[k] = temp[k];
}
```

Figure 6.11: Code

6.8 Bitonic Merge Sort

- Consider an unsorted list with N = 2dim items. Any list with only two items is a bitonic list. Therefore, this unsorted list consists of N/2 bitonic lists of length 2.

- By applying the bitonic merge to pairs of adjacent lists, the result is N/4 bitonic lists of length 4. After logN repetitions of the bitonic merge, the list is completely sorted.

6.9 Parallel Bitonic Merge sort

- **Merge:** Given two ascending sorted sublists, L[0:N-1] and L[N:2N-1], obtain a combined ascending sorted list, L[0:2N]. Consider an unsorted list with N = 2dim items. Any list with only two items is a bitonic list.

- **Recursive Merge Sort:** Given a list, divide it in half, sort the two halves (recursively),and then merge the two list together.

- The Divide and Conquer model is an extended case of the master-worker paradigm where each subtasks might spawn additional tasks. Typically, the higher the task is in the tree, the longer the running time and the more critical is its performability.

- **Bitonic list:**A list with no more than one local maxima and no more than one local minima. One important type of bitonic list has the first half sorted in ascending order and the second half in descending order.

- **Bitonic split:**

    a. Each element in the first half of the list is assigned a partner, which is the same relative position from the second half of the list.

6.10 Reliability Results

6.10.1 Without Reliability Model

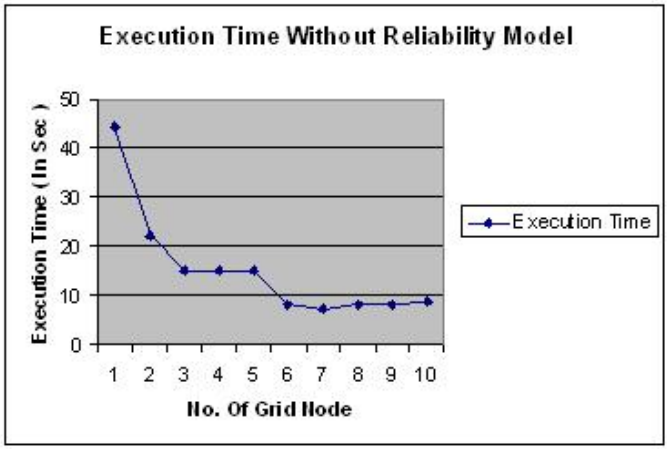

Figure 6.12: Without Reliability Model

6.10.2 With Reliability Model

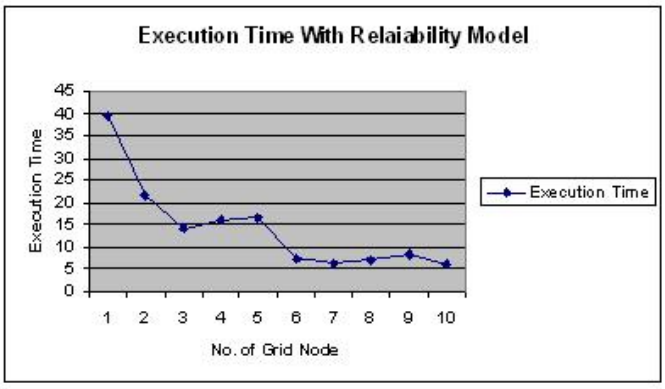

Figure 6.13: With Reliability Model

6.11 Failure To Repair Rate

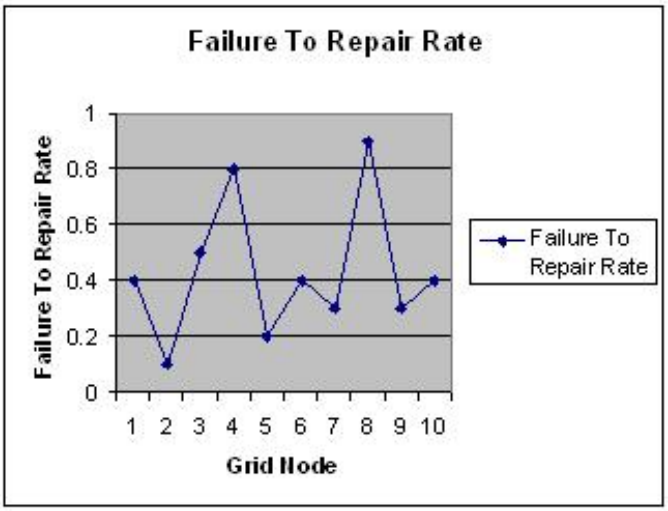

Figure 6.14: Failure To Repair Rate

6.12 Physical Avg.Disk Bytes/write

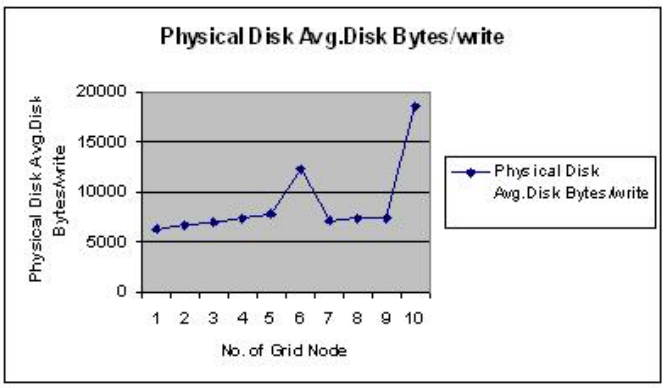

Figure 6.15: Physical Avg.Disk Bytes/write

6.13 Analysis of Results

Showing the results of reliability we can say that using reliability model and programming method we can decrease the execution time of job and increase the performance of grid. Also find the failure to repair rate of each node and decide which node is better for our computation. We use values (from fig.6.14) for the failure-to-repair-rate and performance degradation factors to study the variation in expected steady state reward rates. If we considered only performance, we would pick Grid node 2 as it completes the application most quickly. If we were to select a resource based on reliability, we would pick Grid node 8, the one with the lowest failure-to-repair ratio. Also we can say from fig.6.12 and 6.13 we run without reliability model program the execution time is high as compare to with reliability program and we run with reliability model program the execution time is low as compare to previous one. so we can say that reliability model is very important for decrease the execution time of job.

Chapter 7

Conclusion

7.1 Conclusion

The objective of this project is to deploy the computational grid by applying user driven scheduling policies with an improvement in QoS parameters. Major parameter considered in this project is Time and Reliability.In Major Project,From the results we can say that using Qos based scheduler we can decrease the execution time and increase the performance of grid. QoS based user driven scheduler also provide large scale job by distribute across multiple grid nodes,and also reduce the execution time of a job and increase the performance of the grid. we can find failure to repair rate of each grid node.using reliability model and programming method we can decrease the execution time of job and increase the performance of grid. Also making grid more reliable.

References

[1] Zeljko Stanfel, Goran Martinovic, Zeljko Hocenski,Scheduling Algorithms for Dedicated Nodes in Alchemi Grid,2008 IEEE International Conference on Systems, Man and Cybernetics (SMC 2008)

[2] Fatos Xhafa,Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms,2007 IEEE

[3] Weidong Hao1, Yang Yang1, Chuang Lin2, Zhengli,QoS-aware Scheduling Algorithm Based on Complete Matching of User Jobs and Grid Services, Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06)0-7695-2751-5/06 2006

[4] N.Tonellotto,nicola.tonellotto@isti.cnr.it,R.Yahyapour ramin.yahyapour@unido.de, Ph. Wieder ph.wieder@fz-juelich.de,A Proposal for a Generic Grid Scheduling Architecture

[5] Aline P. Nascimento, Cristina Boeres, Vinod E.F. Rebello Instituto de Computao, Universidade Federal Fluminense (UFF), Niteri, RJ, Brazil depaula,boeres,vinod@ic.uff.br,Dynamic Self-Scheduling for Parallel Applications with Task Dependencies,MGC08 December 1-5, 2008 Leuven, Belgium. Copyright 2008 ACM 978-1-60558-365-5/08/12

[6] Ali Afzal, John Darlington, A. Stephen McGough London e-Science Centre,QoS-Constrained Stochastic Workflow Scheduling in Enterprise and Scientific Grids,1-4244-0344-8/06/2006 IEEE

[7] B.T.Benjamin Khoo et al., A multi-dimensional scheduling scheme in a grid computing environment, Journal of Parallel and Distributed Computing, vol. 67, no. 6, pp. 659-673, June 2007.

[8] S. Ghosh. Distributed Systems: An Algorithmic Approach, Computer and Information Sciences, Chapman & Hall/CRC, 2006.

[9] A. Abraham, R. Buyya and B. Nath. Natures Heuristics for Scheduling Jobs on Computational Grids, The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000) India, 2000.

[10] E. Alba, B. Dorronsoro, and H. Alfonso. Cellular Memetic Algorithms, Journal of Computer Science and Technology, 5(4), 257-263, 2006.

[11] Darshana Shah,Swapnali Mahadik. QoS Oriented Failure Rate-Cost and Time Algorithm for Compute Grid,Department of Computer Engineering and IT

[12] Lavanya Ramakrishnan, Daniel A. Reed. Performability Modeling for Scheduling and Fault Tolerance Strategies for Scientific Workflows, HPDC08, June 2327, 2008, Boston, Massachusetts, USA.

[13] Christopher Dabrowski,Reliability in Grid Computing Systems,National Institute of Standards and Technology,1-4244-0344-8/06/2006 ACM.

[14] Y.S. Dai, M. Xie K.L. Poh,Reliability Analysis of Grid Computing Systems,Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC02) 0-7695-1852-4/02 2002 IEEE..

Index