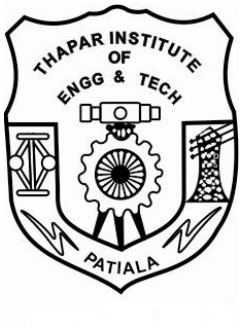
Fault Tolerance in Computational Grids

A Thesis

Submitted in partial fulfillment of the
requirement for the award of degree of

Master of Engineering
In
Software Engineering



Under the Supervision of
Ms. Inderveer Chana
Sr. Lecturer, CSED

By

Inderpreet Chopra
(8043109)

Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
(Deemed University), Patiala-147004 (India).

May 2006

ABSTRACT

The Grid is rapidly emerging as the means for coordinated resource sharing and problem solving in multi-institutional virtual organizations while providing dependable, consistent, pervasive access to global resources. The emergence of computational Grids and the potential for seamless aggregation and interactions between distributed services and resources, has led to the start of new era of computing. Tremendously large number and the heterogeneous nature of grid computing resource make the resource management a significantly challenging job. Resource management scenarios often include resource discovery, resource monitoring, resource inventories, resource provisioning, fault isolation, variety of autonomic capabilities and service level management activities. Out of these scenarios, fault tolerance is one of the main research areas. The probability of fault occurrence increases, as the number of resources involved in grid increases. Till today there is no system that can be fully fault tolerant.

In this research our main focus is on the development of fault tolerance system for computational grids. For this we had setup a computational grid based on the Alchemi middleware. Alchemi is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct computational grid. After setting up grid environment, we have studied existing fault tolerance in Alchemi in detail, and have ascertained the frequent causes of failures in it.

To deal with some of the identified deficiencies we have proposed backup manager concept. Backup manager uses the heartbeating and replication based fault tolerant technique to monitor the central manager. In case of failure of the central manager, backup manager will take its control and avoids the grid to fail.

CERTIFICATE

I hereby certify that the work which is being presented in the thesis titled, **"Fault Tolerance in Computational Grids"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Inderveer Chana*.

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

(***Inderpreet Chopra)***

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

(**Ms. Inderveer Chana)**
Sr. Lecturer
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
Patiala

Countersigned by

(Dr. (Mrs.) Seema Bawa)
Head and Professor, CSED
Thapar Institute of Engg and Tech
Patiala.

(Dr. T.P. Singh)
Dean Academic Affairs
Thapar Institute of Engg and Tech
Patiala.

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my supervisor Ms. Inderveer Chana, Senior Lecturer, Computer Science and Engineering Department, for her immense help, guidance, stimulating suggestions and encouragement all the time with this thesis work. She always provide me motivating and enthusiastic atmosphere to work with, it was a great pleasure to do this thesis under her supervision.

I am equally grateful to Mr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department, a nice person and an excellent teacher, who always encouraged me to keep going with work and always advised me with his invaluable suggestions.

I am highly thankful to Dr. (Mrs.) Seema Bawa, Head and Professor, Computer Science and Engineering Department, for her moral support, cheering concern and her optimistic attitude for doing things. My special thanks goes to Mr. Amit Kumar Bhardwaj, System Analyst, Electronics and Engineering Department, for his great help and assistance in all regards, from technical to personal problems.

I am most indebted to Ms. Anju Sharma, Research Scholar, Computer Science and Engineering Department, for her help, assistance and suggestions during my work. She had always been a great support for work, helped me more than as a mentor, equally like friend with a gesture of family touch.

I would also like to express my appreciation to my colleagues and TIET grid group, it was a great pleasure working with them in this thesis together. And, not to forget the kind assistance of Mr. Balwinder Singh, Lab Supdtt., Computer Science and Engineering Department with any technical problem my PC had. The most valuable thing I acquired from the two years study in TIET is to protect, survive and endure myself in this world with increased confidence and additional faith in my abilities to achieve. Finally, my special thanks go to my family for their never-ending love and support.

Inderpreet Chopra

(8043109)

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION 1

Distributed computing is often used to describe a type of computing in which different computing nodes can simultaneously run an application located on different computers that are connected by a communication network. The main motivation for constructing distributed systems is to obtain large scale resource sharing at affordable cost. To obtain this goal, the interconnection between widely distributed heterogeneous resources has become a necessity for constructing distributed systems. Advances in networking technology and computational infrastructure make it possible to construct large-scale high performance distributed computing environments that provide dependable, consistent and pervasive access to high end computation. Grid computing has been widely seen as a step beyond the conventional distributed computing, incorporating pervasive high-bandwidth, high-speed computing, intelligent sensors and large scale database into a seamless pool of managed and brokered resources. However, the major challenge in such highly heterogeneous and complex computing environment is to design an efficient fault tolerant system. Most of the earlier proposed fault tolerant system were developed for local area networks and are not efficient in the context of wide area distributed systems and are characterized by a high level of asynchrony, long message delay and high probability of message loss. The large number of components and their distribution in grid as well as the dynamic structure of the grid increases the risk of failures. Thus providing a scalable and generic fault tolerant system as a basic service for Grids is of primary importance in such systems. In this chapter, we identify the research questions and define the motivation of this research. Then, we present brief introduction of the project and finally give the framework of this thesis.

1.1 MOTIVATION AND RESEARCH QUESTIONS

Grid Computing enables aggregation and sharing of geographically distributed computational, data and other resources as single, unified resource for solving large-scale compute and data intensive computing application. Management of these resources is an important infrastructure in the grid computing environment. It becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access and cost models, and have dynamically varying loads and availability. Due to highly heterogeneous and complex computing environments, the chances of faults increases, therefore it is necessary to design a mechanism to check for faults and handle them efficiently in such infrastructure. Handling faults (configuration, middleware, application and hardware faults) in highly dynamic networks where the availability of resources change at a high rate, requires a mechanism that is scalable, adaptable, robust and reliable.

The conventional resource management schemes are based on relatively static model that have centralized controller that manages jobs and resources accordingly. These management strategies might work well in those scheduling regimes where resources and tasks are relatively static and dedicated. However, this fails to work efficiently in many heterogeneous and dynamic system domains like Grid where jobs need to be executed by computing resources, and the requirement of these resources is difficult to predict. The complex, heterogeneous and dynamic systems present new challenges in resource management such as: scalability, adaptability, reliability and fault tolerance. This poses numerous research questions like:

- Grid is a dynamic environment where location, type and performance of components are constantly changing; hence their computational requirement varies over time. How to adapt to such conditions when computational capacities fluctuate in high ratio? One way to adapt to the changes as the availability of resource may fluctuate due to connection/disconnection of computing resources is to use autonomous entities.

- In Grid environment, computing nodes can join and leave the system at any time without any dedicated commitment, another issue concerned for GRID management is how to tolerate such failures and recover from them when crashes occur? How to re-allocate task and resource automatically? One approach for solving this problem is to use robust entities.
- How to manage the intricacy, complexity, performance analysis in such dynamic system where nothing is predictable? For manageability it is useful to use intelligent entities.
- What are the more frequent kinds of failures you face on Grids?
- What are the mechanisms used for detecting and/or correcting and/or tolerating faults?
- What are the greatest problems you encounter when you need to recover from a failure scenario?
- To what degree is the user involved during the failure recovery process?
- What are the greatest users' complaints?
- Are there mechanisms for application debugging in your grid environment?

1.2 THESIS CONTRIBUTIONS

Contribution of this thesis is as follows:

- The first contribution of this thesis is towards setting up a grid that create a dynamic network accessible computing resources, such as processing power, memory storage, disk space, etc., in a completely distributed, scalable, and fault-tolerant manner. For setting up the grid we have used Alchemi, a .Net based middleware from the Gridbus project.
- The second contribution of this thesis is to identify and build fault tolerance mechanisms in the grid setup. For this, we propose a new fault tolerance system for the setup grid.
- The third contribution of this thesis is .Net based implementation of proposed fault tolerance system,

1.3 THESIS FRAMEWORK

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

Chapter 2 reviews the background of this thesis from two aspects: First, it discusses about beginnings of the grid its motivation, issues and characteristics. Second, it presents the Open Grid Service Architecture (OGSA) and the Resource Management for grids.

Chapter 3 reviews and analyzes need for fault tolerance in heterogeneous and dynamic environment such as Computational Grids. First part defines the Computational Grids, their characteristics and their main design features. Second, it summaries the kind of faults that can occur in grids and various ways to deal with these kind of faults. Also a survey describing fault tolerance mechanism in various grid middleware is also defined.

Chapter 4 gave the project description. In this we first introduce the Alchemi as the computational grid and then gave reasons for choosing Alchemi as the framework for this thesis. The second part of this chapter discusses the Fault Tolerance in Alchemi- What is there for fault tolerance and what still lacks in Alchemi for fault tolerance.

Chapter 5 discusses about the purpose and implementation of the new fault tolerance system for the Alchemi based Computational grids. This purposed system will try to deal with all the deficiencies present in Alchemi.

Chapter 6 presents the conclusion of the thesis, describes the main contribution of the thesis and highlights future research direction based on the results obtained.

GRID: AN OVERVIEW 2

The last decade has seen a considerable increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. Nevertheless, there are still problems, in the fields of science, engineering and business, which cannot be dealt effectively with the current generation of supercomputers. In fact, due to their size and complexity, these problems are often numerically and/or data intensive and require a variety of heterogeneous resources that are not available from a single machine. A number of teams have conducted experimental studies on the cooperative use of geographically distributed resources conceived as a single powerful computer. This new approach is known by several names, such as, metacomputing, seamless scalable computing, global computing, and more recently grid computing. The early efforts in grid computing started as a project to link supercomputing sites, but now it has grown far beyond its original intent. In fact, there are many applications that can benefit from the grid infrastructure, including collaborative engineering, data exploration, high throughput computing, and of course distributed supercomputing. Moreover, the rapid and impressive growth of the Internet, there has been a rising interest in web-based parallel computing. In fact, many projects have been incepted to exploit the Web as an infrastructure for running coarse-grained distributed parallel applications. In this context, the web has the capability to become a suitable and potentially infinite scalable metacomputer for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous grid infrastructure.

This chapter is organized as follows: First few sections Section 2.1-2.6 review the beginnings of the grid and talk about grid computing its issues and why we need it. Section 2.7 defines the computational grids and in next section we discuss Open Grid Service Architecture (OGSA), standard given by GGF for forming the standard grid. Section 2.9 discusses the Resource Management and its challenges. From the challenges we pick Fault Tolerance and explain it in brief in section 2.10.

2.1 BEGININGS OF THE GRID

Many of the basic ideas behind the Grid have been around in one form or other throughout the history of computing. For example, one of the "novel" ideas of the Grid is sharing computing power. Nowadays, where most people have more than enough computing power on their own PC, sharing is unnecessary for most purposes. But back in the sixties and seventies, sharing computer power was essential [1]. At that time, computing was dominated by huge mainframe computers, which had to be shared by whole organizations.

In 1965 the developers of an operating system called Multics (an ancestor of Unix, which in turn is an ancestor of Linux - a popular operating system today) presented a vision of "computing as a utility" - in many ways uncannily like the Grid vision today. Access to the computing resources was envisioned to be exactly like water, gas and electricity - something which the client connects to and pays for according to the amount of use. Ironically, "utility computing" is all the rage again these days, and used more or less as a synonym for the Grid by some people.

So, yes, there is a certain amount of "reinventing the wheel" going on in developing the Grid. However, each time the wheel is reinvented, it is reinvented in a much more powerful form, because computer processors, memories and networks improve at an exponential rate which is associated with Moore's law.

Because of the huge improvements of the underlying hardware (typically more than a factor of 100x every decade), it is fair to say that reinvented wheels are qualitatively different solutions, not just small improvements on their predecessor

2.2 ANCESTORS OF THE GRID

Although the Grid can trace its intellectual roots back to the operating system Multics, and no doubt even further back into computing history, the immediate ancestor of the Grid is "metacomputing", a term that dates back to around 1990, and was used to describe projects aimed at interconnecting supercomputer centers in the US in order to combine the processing power of multiple supercomputers.

Two cutting edge metacomputing projects in the US, both conceived in 1995, were called **FAFNER** [2] and **I-WAY** [3]. Each in its own way influenced the evolution of some of the key Grid technology projects ongoing today.

FAFNER (Factoring via Network-Enabled Recursion) was a project which aimed at factorizing very large numbers, a computationally intensive challenge which is very relevant to digital security. Since this is a challenge which can be broken into small parts, even fairly modest computers can contribute useful power. Many of the techniques developed for splitting up and distributing a big computational problem were forerunners of the technology used for SETI@home and other "cycle scavenging" software.

I-WAY (Information Wide Area Year) was a project that aimed at linking supercomputers, but using only existing networks, not building new ones. One of the innovations of the I-WAY project was developing a computational resource broker, conceptually very similar to the Resource Brokers being developed for the Grid today. I-WAY strongly influenced the development of the Globus Project, which is at the core of most Grid activities, as well as the LEGION project, an alternative approach to distributed supercomputing.

The Grid, itself, was born at a workshop at Argonne National Laboratory in September 1997, called "building a computational Grid". In the late 1990s, Grid researchers came together in the Grid Forum, subsequently expanding to the Global Grid Forum (GGF) [4], where much of the early research is now evolving into the standards base for future Grids. Recently, the GGF has been instrumental in the development of the Open Grid Services Architecture (OGSA) [5], which integrates Globus and Web services approaches. OGSA is being developed by both the United States and European initiatives aiming to define core services for a wide variety of areas including:

- *Systems Management and Automation*
- *Workload/Performance Management*
- *Security*
- *Availability/Service Management*

• *Logical Resource Management*
• *Clustering Services*
• *Connectivity Management*
• *Physical Resource Management*.

Today, the Grid has gone global, with much worldwide collaboration between the United States, European and Asia-Pacific researchers. Funding agencies, commercial vendors, academic researchers, and national centers and laboratories have come together to form a community of broad expertise with enormous commitment to building the Grid. Moreover, research in the related areas of networking, digital libraries, peer-to-peer computing, collaboratories and so on are providing additional ideas relevant to the Grid.

2.3 KEY CONCEPTS

There are a number of technologies that evolved during the 90s, which make the technological environment of the Grid today substantially different from what was around when metacomputing emerged in 1990. It's worth listing some of these, if only to remind oneself just how fast the world of computing changes:

1990 The World Wide Web: Tim Berners Lee began work at CERN on the Web in that year, culminating in CERN releasing the Web to the world in 1993. The web plays a crucial role in simplifying communications on the Grid.

1991 The Linux Operating System: Linus Torvalds, then at the University of Helsinki, began work in 1991 on this open source operating system, which has an army of thousands of unpaid programmers ensuring that it is "tweaked" to work optimally on every new piece of hardware that comes along. This is one reason why Linux is the preferred Operating System for many Grid testbeds.

1994 Beowulf clusters of PCs: The first cluster of cheap "commodity" computers that was put together by Donald Becker and Thomas Sterling at NASA in 1994. They used Ethernet cards to provide high speed links between the computers, in order to emulate the sort of computing power previously only available from expensive supercomputers. Now such clusters are commercially available, already assembled,

from most PC manufacturers. The Grid, at least for most scientific purposes, will rely on such clusters.

1995 Java programming language: This programming language was developed by software engineers at Sun Microsystems to be independent of the computer it is running on (initially it was meant for use in consumer appliances, not computers). Although Java is not the only computer language used for programming the Grid, it is an extremely popular one, and well-adapted to the Grid philosophy.

2.4 BACK TO THE FUTURE

To get a feeling of just how much more powerful the Grid is than its intellectual ancestor, metacomputing, at the beginning of the nineties, consider the following facts [6].

A typical PC today is as powerful as a giant supercomputer was a decade ago. PCs are shipped with over 100 Gigabytes of memory, as much as entire computer centres could muster at the beginning of the 90s. And ADSL links from people's homes to the internet typically have speeds well in excess of the 56kbits/sec which was the state-of-the-art for connectivity between major supercomputer centres in 1985.

Now consider clusters of hundreds of PCs, linked together to form computing engines that were unimaginable just a decade ago. And consider linking such clusters around the Globe with Gigabit/sec wide-area networks, which are available today, and are more than 1000x faster than what was available in 1990. If you had described this sort of system - which is what the Grid aims to be today - to someone back in 1990, it would have been about as mind-boggling as telling the Wright brothers about a Jumbo Jet!

Now consider that you would probably have exactly the same shock if someone from 2020 came back to visit you and tell you what computing will be like by then. That certainly puts the Grid in a historical perspective!

2.5 WHAT IS GRID?

The short answer is that, whereas the Web is a service for sharing information over the Internet, the Grid is a service for sharing computer power and data storage

capacity over the Internet. The Grid [7] goes well beyond simple communication between computers and aims ultimately to turn the global network of computers into one vast computational resource.

That is the dream. But the reality is that today, the Grid is a "work in progress", with the underlying technology still in a prototype phase, and being developed by hundreds of researchers and software engineers around the world.

The Grid is attracting a lot of interest because its future, even if still uncertain, is potentially revolutionary. We can say that *Grid computing* [8] is the process of using resources of many computers in a network to a single problem at a time. The use of unexploited resources of distant machines considerably increases computational power, enhances data storage, data recovery and supplies with further facilities.

2.6 WHY GRID COMPUTING?

Computers have been proven to be very efficient to solve complex scientific problems. They are used to model and simulate problems of a wide range of domains; for instance medicine, engineering, security control and many more. Although their computational capacities have shown greater capabilities than the human brain to solve such problems, computers are still used less than they could be. One of the most important reasons to this lack of use of computational power is that, despite the relatively powerful computing environment one can have, it is not adapted to such complicated computational purposes.

2.6.1 EXPLOITING UNUSED RESOURCES

In most organizations, computing resources are underutilized. Most desktop machines are busy less than 25% of the time (if we consider that a normal employee works 42 hours a week and that there are 168 hours per week) and even the server machines can often be fairly idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage [9].

The easiest use of Grid computing would be to run an existing application on several machines. The machine on which the application is normally run might be unusually busy, the execution of the task would be delayed. Grid Computing should enable the

job in question to be run on an idle machine elsewhere on the network. From this point of view, Grid Computing looks like a great answer to numerous problems without prodigious management requirements. Convenience does not always imply simplicity and grid enabled applications have to meet two prerequisites: first, the application must be executable remotely and with low additional operating cost; second, the remote machine must meet any special hardware, software, or resource requirements needed to run the application. Hence, remote computing is not only a matter of sending jobs to distant machines but also choosing the correct machine that has all the required software and hardware.

Even though Computing Power (CPU) can be assumed to be the most commonly shared resource, it belongs to a large set of underutilized resources that are to be shared on the network such as storage capacity, software, internet connection and the like which also need to be considered.

2.6.2 INCREASE IN COMPUTATION

To provide users with more computational power [10], some crucial areas have to be considered:

- Hardware Improvement: microprocessor architecture and other resource capabilities of personal computers continuously increase to provide users with additional power.
- Periodic computational needs: some applications only need computational power once in a while. The systems is fully utilized at times and idle the rest of the time.
- Capacity of Idle Machines: machines are often idle and thus their computational power is free to use. The idea would be to use it only during that idle time and leave once the computer is in use.
- Sharing of Computational results: the key to more sharing may be the development of collaboratories: "...centers without walls, in which the nation's researchers can perform their research without regard to geographical location-interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries".

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

2.7 COMPUTATIONAL GRID

Computational Grid is a collection of distributed, possibly heterogeneous resources which can be used as an ensemble to execute large-scale applications. Computational Grid is also called *metacomputer* .Term computational Grid comes from an analogy with the electric power Grid [11].

"*A computational Grid is hardware and a software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities*."

This definition include some of the specific words and they have there specific meaning concerned to definition, these meanings are

Dependable service means assurance to user that they will receive predictable, sustained, and often high levels of performance from the diverse components that constitute the Grid in the absence of such assurances; applications will not be written or used.

The need for **consistency** of service is a second fundamental concern. As with electric power, we need standard services, accessible via standard interfaces, and operating within standard parameters. Without such standards, application development and pervasive use are impractical.

Pervasive access allows us to count on services always being available, within whatever environment we expect to move. Pervasiveness does not imply that resources are everywhere or are universally accessible. We cannot access electric power in a new home until wire has been laid.

Finally, an infrastructure offers **inexpensive** (relative to income) access if it is to be broadly accepted and used. Homeowners and industrialists both make use of remote billion- dollar power plants on a daily basis because the cost to them is reasonable.

2.8 OGSA: OPEN GRID SERVICES ARCHITECTURE

The objectives of OGSA [12] are to provide a common base for autonomic management solutions to manage heterogeneous resources when necessary; define open and published interfaces for resource interoperability; manage heterogeneous resources across heterogeneous platforms; meet with QoS requirements; exploit integration technologies in place.

OGSA interfaces address discovery, dynamic service creation, lifetime management, notification and manageability whereas conventions address naming and upgradeability. They both are concerned, in particular, with behaviors related to the management of transient service instances. OGSA can be further extended into OGSA.NET, this basically deals with .NET based grid environments. Here we are only explaining OGSA, for OGSA.NET refer to Appendix B.

2.8.1 OGSA ARCHITECTURE

We introduce the OGSA architecture [5] divided in four main layers (Figure 2.1): Resources (physical or logical), Web services (OGSI), OGSA services and Grid applications.

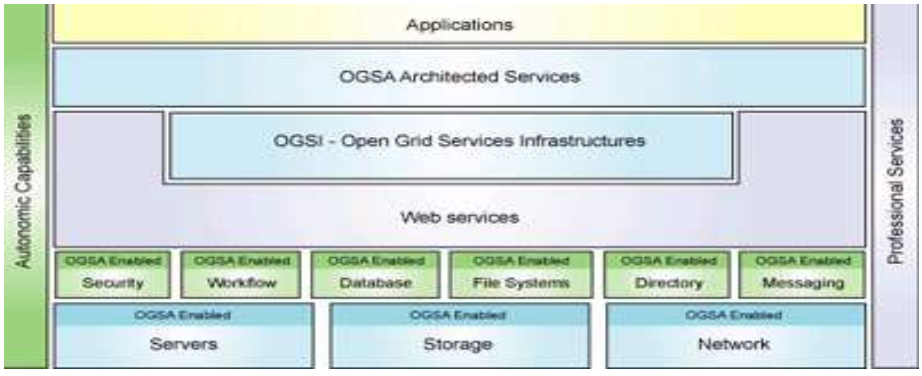


Figure 2.1: OGSA Layered Architecture [11]

2.8.1.1 OPEN GRID SERVICE INFRASTRUCTURE LAYER (OGSI)

The Grid infrastructure is built on top of Web services standards, hence leveraging the great effort -in terms of tools and support- that the industry has put into the field.

OGSI exploits the mechanisms of Web services like XML and WSDL to specify standard interfaces, behaviors, and interaction for all grid resources. Nevertheless, some key characteristics of Web services standards are missing such as ability to create new services on demand (the Factory pattern); service lifetime management; statefulness; access to state; notification; and service groups. OGSI is based on the concept of a Grid service instance and all services in the OGSA platform must adhere to the conventions specified by OGSI and, therefore, they all are Grid services (Figure 2.2).

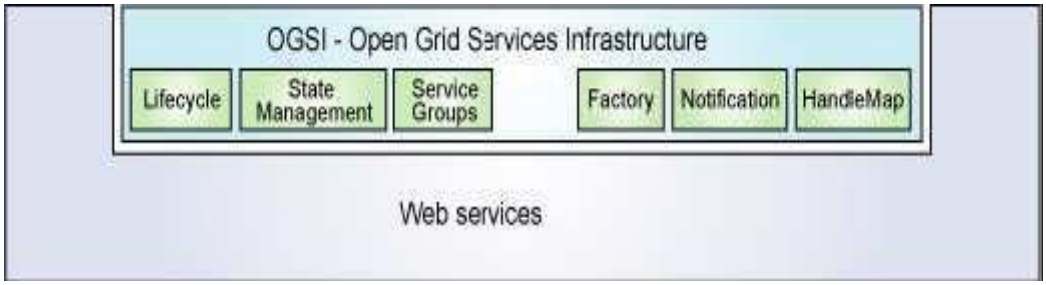


Figure 2.2: OGSI Services [11]

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

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

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

Service groups Service data indexes groups of grid services, Service groups, for some specific purpose. For example, they might be used to collect all the services that represent the resources in a particular cluster-node within the grid.

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

HandleMap When factories are used to create a new instance of a grid service, the factory returns the identity of the newly instantiated service. The result of this request is a Grid Service Handle (GSH) and a Grid Service Reference (GSR). A GSR is created with finite lifetime during which it might change while a GSH is guaranteed to reference the grid service indefinitely [17]. The HandleMap interface provides a way to obtain a GSR given a GSH. First, we need to identify the HandleMap service that contains the mapping for the precise GSH and then contact this specific HandleMap to get the required GSR.

2.8.1.2 PHYSICAL AND LOGICAL RESOURCES LAYER

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

2.8.1.3 OGSA ARCHITECTED GRID SERVICES LAYER

The Web services layer, with its OGSI extensions, provide a base infrastructure for the next layer – architected grid services. We have seen that OGSI was an important step towards the development of a Grid SOA. However, a wide collection of grid services need to be implemented and provided by both middleware software and grid software to ensure application development. This additional layer can be subdivided into four categories (Figure 2.3):

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

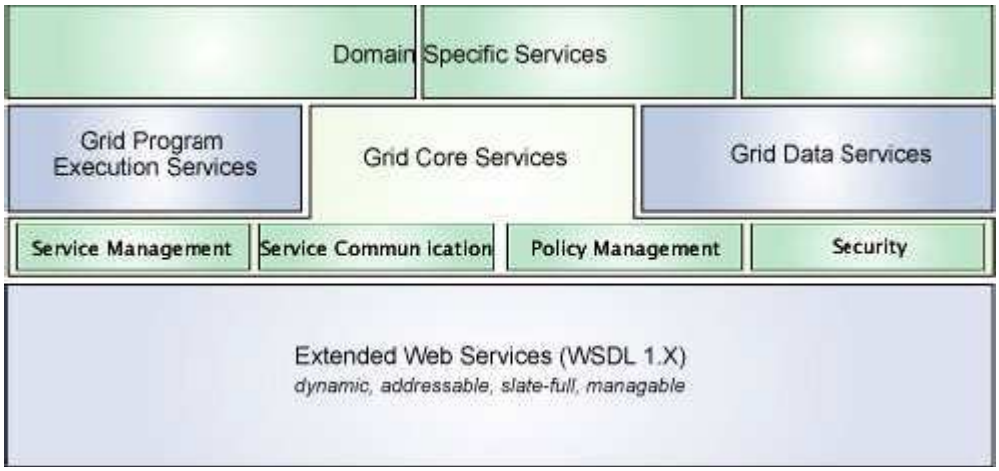


Figure 2.3: Grid Core Services [11]

We can have a closer look at these so that we understand what the role of this layer, as a whole, is**. Grid Core Services** We see in Figure 2.3, that the grid core services are dividable in four main types of grid services:

- Service management
- Policy management

- Service communication
- Security

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

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

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

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

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

Grid Program Execution Services

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

Grid Data Services

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

2.8.1.4 GRID APPLICATIONS LAYER

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

2.9 RESOURCE MANAGEMENT

Resource management is the process of managing available resources and system workloads accordingly [13]. Resource management becomes more important in the case where some resources are idle and could be used to relieve overloaded nodes in the same network. There are basically two principal reasons to address resource management in distributed system. First, available system resources should be maximally (or optimally) used in order to balance the distribution of the tasks over the different computing nodes. Second, when events such as local congestion, a node failure or communication failure occurs, it is necessary to manage (re-)allocation jobs and resources accordingly [14].

Resource management is a complex task involving security and fault tolerance along with scheduling. It is the manner in which resources are allocated, assigned, authenticated, authorized, assured, accounted, and audited. Resources include traditional resources like compute cycles, network bandwidth, space or a storage system and also services like data transfer, simulation etc.

Classification of RMS is based on grid type, resource namespace, resource information (discovery, dissemination), scheduling model and scheduling policy [15].

2.9.1 GRID TYPE

Grid systems are classified as Compute, Data and Service grids as shown in figure. The computational Grid category denotes the systems that have a higher aggregate computational capacity available for single applications than the capacity of any constituent machine in the system.

In Data Grids the resource management system manages data distributed over geographical locations. Data Grid is for systems that provide an infrastructure for synthesizing new information from data repositories such as digital libraries or Data Warehouses that are distributed in a wide area network.

The Service Grid category is for the systems that provide services that are not provided by any single machine. This category is further subdivided in On Demand, Collaborative, and Multimedia Grid Systems.

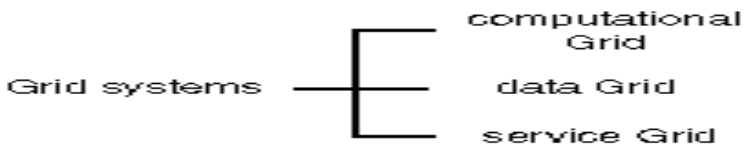


Figure 2.4: Grid Type [13]

2.9.2 RESOURCE NAMESPACE

Resources in a grid are managed and named by the Grid Resource Management System; the naming of resources effects others functions of GRMS [16] like resource discovery, resource dissemination and also affects the structure of the database storing resource information. Different approaches to naming are Relational, Hierarchical and graph based.

A relational namespace divides the resources into relations and uses the concepts from relational databases to indicate relationships between tuples in different relations.

A hierarchical namespace divides the resources in the grid into hierarchies organized around the physical or logical network structure of the grid i.e. it follows a system of systems approach, a name is constructed by traversing down a hierarchy.

In Graph Based Naming resources are linked together and a resource name is constructed by following the links from one object to another.

2.9.3 RESOURCE INFORMATION

Resource Information is further characterized into two sub sections. These are:

2.9.3.1 RESOURCE DISSEMINATION

Resource dissemination is the process of advertising information about resources. The protocols used for dissemination are "periodic" and "on demand". In periodic resource dissemination the information database is updated periodically so update is not driven by resource status change indeed all changes are batched and updated in information database after specific interval. On Demand protocol updates the resource information database as the change occurs in the status of any of the resource.

2.9.3.2 RESOURCE DISCOVERY

Resource management system performs resource discovery to obtain information about available resources. There are two approaches to resource discovery namely "agent based" and "query based". In Agent based approach agents traverse the grid system to gather information about resource availability. In Query Based approach resource information store is queried for resource availability.

2.9.4 SCHEDULING MODEL

Scheduling model describes how machines involved in resource management make scheduling decisions. Scheduling models normally used are centralized and decentralized; in a centralized model all jobs are submitted to a single machine which is responsible for scheduling them on available resources. The problems with this approach are that the single scheduler will be single point of failure. It will also affect scalability of the grid. In decentralized model there is no central scheduler, scheduling is done by the resource requestors and owners independently. This approach is scalable and suits grid systems. But individual schedulers should cooperate with each other in making scheduling decisions.

2.9.5 SCHEDULING POLICY

Scheduling policy governs how resources are scheduled on the matched resources. In a Grid environment there can be no single global scheduling policy, Different

administrative domains may set different resource usage policies, so the RMS should allow for the policies to be added or changed with minimal overhead.

2.9.6 CHALLENGES IN RESOURCE MANAGEMENT

Various challenges encountered while managing the resources in grids are [17]:

- Partitioned large complex allocation problems into smaller, disjoint allocation problems
- Decentralized resource access, allocation and control mechanism
- Design reliable, fault tolerant and robust allocation mechanism
- Design scalable architecture for resource access in a complex system provides guarantees to users and applications on performance criteria. Some of the performance criteria in distributed systems include the following:
    - average response time
    - throughput
    - user access time
    - message delay time
    - information loss
    - application failure probability
- Define system performance criteria that reflect in aggregate the diverse individual criteria of users and applications
- Design a unified framework in which users have transparent access to the services of a distributed system, and services are provided in as efficient manner. This framework should hide the complexity or multiple resource suppliers and resource allocation policies.

Among all these challenges we are going to deal with the challenge of fault tolerant system. Making a fault tolerant system for distributed systems like grid is a very challenging task. In the coming chapters we will see why dealing with faults is a challenging process.

2.10 FAULT TOLERANCE

As a result of the complex nature of heterogeneous networks, fault tolerance is a major concern for the network administrators, and there are various ways that detection of such occurrences can be accomplished. When a fault occurs, it is important to:

- rapidly determine exactly where the fault is,
- Isolate the rest of the network from the failure so that it can continue to function without interference,
- Reconfigure or modify the network in such a way as to minimize the impact of operation without the failed component or components, and
- Repair or replace the failed components to restore the network to its initial state.

As grid is a distributed and unreliable system involving heterogeneous resources located in different geographical domain, for this case fault tolerant resource allocation services [14] have to be provided. In particular, when crashes occur, tasks have to be reallocated quickly and automatically, in a completely transparent way from the user's point of view.

One of the main challenges for grid computing is the ability to tolerate failure and recover from them (ideally in a transparent way). Current grid middleware still lacks mature fault tolerant features and next generation grids needs to solve this problem providing a more dependable infrastructure to execute large-scale computations by using remote clusters and high performance computing system.

2.11 CONCLUSION

In this chapter we have studied the introduction to grid computing. We started with the beginnings of the Grid and tried to understand the meaning of Grid Computing. We then defined the computational grids. For making the standard Grid, GGF has given a standard is called Open Grid Services Standard (OGSA).The four layered architecture for OGSA has been discussed. As grid consists of large number of heterogeneous resources that kept on varying dynamically, the management of resources is not an easy job. Hence Resource Management System (RMS) plays very

important role in grid computing. Classification of RMS can be done on the basis of: grid type, resource namespace, resource information (discovery, dissemination), scheduling model and scheduling policy. Large numbers of challenges are in front of RMS and each constitutes different research area. Among these research areas Fault Tolerance is emerging area.

In the next chapter we are going to discuss Fault Tolerance in Computational Grids in more detail.

FAULT TOLERANCE TECHNIQUES 3

Computational Grids have the potential to become the main execution platform for high performance and distributed applications. However, such systems are extremely complex and prone to failures. The use of computational grids as a platform to execute parallel applications is a promising research area.

The possibility to allocate an enormous amount of resources to a parallel application (thousands of machines connected through the Internet) and to make it with lower cost than traditional alternatives (based in parallel supercomputers) is one of the main attractive in grid computing.

Grid characteristics, as high heterogeneity, complexity and distribution – traversing multiple administrative domains – create many new technical challenges, which need to be addressed.

In particular, grids are more prone to failures than traditional computing platforms. In a grid environment there are potentially thousands of resources, services and applications that need to interact in order to make possible the use of the grid as an execution platform. Since these elements are extremely heterogeneous, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them.

In this chapter is organized as follows: Section 3.1 discusses the computational grids, there features and the factors to be considered while designing the computational grids. Section 3.2 describes the fault detection, correction and avoidance mechanisms and the problems that are to be dealt with using the fault tolerance system. Section 3.3 & 3.4 describes the kinds of faults that can occur in grids and approaches to handle them. Section 3.5 gave the surveys to capture the actual experience regarding the fault treatment in different middlewares.

3.1 COMPUTATIONAL GRIDS

Foster et al. have identified three approaches to building computational grids: the commodity approach, the service approach, and the integrated architecture approach [18]. In the *commodity approach*, existing commodity technologies, e.g., HTTP, CORBA, COM, Java, serve as the basic building blocks of the grid. In the *service approach*, as exemplified by the Globus project, a set of basic services such as security, communication, and process management are provided and exported to developers in the form of a toolkit. In the *integrated architecture approach*, resources are accessed through a uniform model of abstraction. There are three main issues that characterize computational grids:

- Heterogeneity: a grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.
- Scalability: a grid might grow from few resources to millions. This raises the problem of potential performance degradation as a Grids size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant.
- Dynamicity or Adaptability: in a grid, a resource failure is the rule, not the exception. In fact, with so many resources in a Grid, the probability of some resource failing is naturally high. The resource managers or applications must tailor their behaviour dynamically so as to extract the maximum performance from the available resources and services.

The following are the main design features required by a grid environment [19]:

- Administrative Hierarchy - An administrative hierarchy is the way that each grid environment divides itself up to cope with a potentially global extent. The administrative hierarchy determines how administrative information flows through the grid.
- Communication Services - The communication needs of applications using a grid environment are diverse, ranging from reliable point-to-point to unreliable multicast communications. The communications infrastructure needs to support protocols that are used for bulk-data transport, streaming data, group

communications, and those used by distributed objects. The network services used also provide the grid with important Quality of Service parameters such as latency, bandwidth, reliability, fault-tolerance, and jitter control.

- Information Services - A grid is a dynamic environment where the location and type of services available are constantly changing. A major goal is to make all resources accessible to any process in the system, without regard to the relative location of the resource user. It is necessary to provide mechanisms to enable a rich environment in which information about grid is reliably and easily obtained by those services requesting the information.
- Naming Services – In a grid, like in any distributed system, names are used to refer to a wide variety of resources such as computers, services, or data objects. The naming service provides a uniform name space across the complete metacomputing environment. Typical naming services are provided by the international X.500 naming scheme or DNS, the Internet's scheme.
- Distributed File Systems and Caching – Distributed applications, more often than not, require access to files distributed among many servers. A distributed file system is therefore a key component in a distributed system. From an applications point of view it is important that a distributed file system can provide a uniform global namespace, support a range of file I/O protocols, require little or no program modification, and provide means that enable performance optimizations to be implemented, such as the usage of caches.
- Security and Authorization – Any distributed system involves all four aspects of security: confidentiality, integrity, authentication and accountability. Security within a grid environment is a complex issue requiring diverse resources autonomously administered to interact in a manner that does not impact the usability of the resources or introduces security holes/lapses in individual systems or the environments as a whole.
- System Status and Fault Tolerance – To provide a reliable and robust environment it is important that a means of monitoring resources and applications is provided. To accomplish this task, tools that monitor resources and application need to be deployed.

- Resource Management and Scheduling – The management of processor time, memory, network, storage, and other components in a grid is clearly very important. The overall aim is to efficiently and effectively schedule the applications that need to utilize the available resources in the metacomputing environment. From a user's point of view, resource management and scheduling should be transparent; their interaction with it being confined to a manipulating mechanism for submitting their application.
- Programming Tools and Paradigms – Grid applications (multi-disciplinary applications) couple resources that cannot be replicated at a single site even or may be globally located for other practical reasons. A grid should include interfaces, APIs, utilities and tools so as to provide a rich development environment. Common scientific languages such as C, C++, and Fortran should be available, as should application-level interfaces like MPI and PVM.
- User and Administrative GUI – The interfaces to the services and resources available should be intuitive and easy to use. In addition, they should work on a range of different platforms and operating systems. They also need take advantage of web technologies to offer a view of portal supercomputing. The web centric approach to access supercomputing resources should enable users to access any resource from anywhere over any platform at any time. The provision of access to scientific applications through the web leads to the creation science portals.

3.2 FAULT TOLERANCE

The promise of Computational Grids for effectively harnessing computing resources across organizations is enormous. The Grid research community has made great strides in defining middleware for core job management and security services. In a grid environment there are potentially thousands of resources, services and applications that need to interact in order to make possible the use of the grid as an execution platform. Since these elements are extremely heterogeneous, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them. Because of the inherent instability of grid environments, fault-detection and recovery is another critical component that must be addressed. The need for fault-tolerance is especially acute for large parallel

applications since the failure rate grows with the number of processors and the duration of the computation. Fault tolerance is the survival attribute of computer systems. The function of fault tolerance is [20]

"...to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service."

Fault tolerance is the property of a system that continues operating consistent with its specifications even in the event of failure of some of its parts [21]. From a user's point of view, a distributed application should continue despite failures. The fault tolerance has become the main topic of research. Till now there is no single system that can be called as the complete system that will handle all the faults in grids. Grid is a dynamic system and the nodes can join and leave voluntarily. For making fault tolerance system a success, we must consider:

- how new nodes join the system,
- how computing resources are shared,
- how the resources are managed and distributed

While considering all these factors, we have to consider the desirable properties of failure detectors [22] and correctors that are summarized as follows:

- Completeness: Any failure (process or machine crash) is eventually detected by all normal processes.
- Accuracy: The probability of false positives is low.
- Consistency: All processes that are not declared as failed obtain consistent failure information including false positives.
- Detection Latency: The latency between a failure and its detection is low.
- Scalability: CPU and network loads are low for a large number of participating processes.
- Flexibility: Various network settings are tolerated. In particular, firewall or NATs may restrict connectivity between some live node pairs. Finally, they must be accomplished without tedious and error-prone manual configuration.

- Adaptiveness: Systems should bring up with a small amount of manually-supplied information about network settings.
- Low overhead. Monitoring should not have a significant impact on the performance of application processes, computers, or networks.

3.3 KINDS OF FAILURES

There can be many different reasons that can lead to the fault occurrence in grids. Some of the reasons we are able to find are as follows:

3.3.1 HARDWARE FAULTS

Hardware failures take place due to faulty hardware components such as CPU, memory, and storage devices. Sometime hardware fails because components are used beyond their specifications [18]. Failure rates of hardware components are low and still decreasing. Moreover, hardware performance degrades after two to three years.

- CPU faults arise due to faulty processors, resulting in incorrect output.
- Memory faults are errors that occur due to faulty memory in the RAM, ROM or cache.
- Storage faults occur for instance in secondary storage devices with bad disk sectors.
- Components that are used beyond specification. Many system components are designed to function within a given operating condition such as temperature, load or even maximum continuous operating time. For example, a computer system specified to work perfectly between 25-30^{O} C room temperature, but otherwise the computer system is prone to failure.

3.3.2 APPLICATION AND OPERATING SYSTEM FAULTS

Application and operating system failures occur due to application or operating system specific faults.

- Memory leaks are an application specific problem, where the application consumes a large amount of memory and never releases it.
- Operating system faults include deadlock or inefficient or improper resource management.

- Resource unavailable. Sometimes an application fails to execute because of resource unavailability, that is, the need for a resource that is in use by other applications.

3.3.3 NETWORK FAULTS

In a grid, computing resources are connected over multiple and different types of distributed networks. As a result, physical damage or operational faults in the network are more likely [23]. The network may exhibit significant packet loss or packet corruption. Moreover, individual nodes in the network or the whole network may go down.

- Node failure. In node failure, individual nodes may go down due to operating system faults, network faults or physical damage.
- Packet loss. Broken links or congested networks may result in significant packet loss.
- Corrupted packet. Packets can be corrupted in transfer from one end to another.

3.3.4 SOFTWARE FAULTS

There are several high resource intensive applications running on grid to do particular tasks. Several software failures like the following can take place while running this software application.

- Un-handled exception. Software faults occur because of un-handled exception like divide by zero, incorrect type casting etc.
- Unexpected input. Operators may enter incorrect or unexpected values into a system that causes software faults, for example specifying an incorrect input file or invalid location of an input file. David Patterson et al in shows that more than 50% of errors take place due to operator slips or lapse, where people do not do what they were expected to do.

3.3.5 RESPONSE FAULTS

Several kinds of response faults like the following can occur:

- Value fault. If some lower level system or application level fault has been overlooked (or due to some unknown problem) an individual processor or application may emit incorrect output.
- Byzantine error. Byzantine errors take place due to failed or corrupted processors that behave arbitrarily.
- Timeout faults. Timeout faults are higher level faults that take place due to some lower level faults at the hardware, operating system and application, software or network level.

3.4 FAULT TREATMENT MECHANISMS

In addition to ad-hoc mechanisms – based on users complaints and log files analysis – grid users have used automatic ways to deal with failures in their Grid Environment. To achieve the automatic ways to deal with failures, various fault tolerance mechanisms are there. Some of these fault tolerance mechanisms are:

- **Application-dependent:** Grids are increasingly used for applications requiring high levels of performance and reliability, the ability to tolerate failures while effectively exploiting the resources in scalable and transparent manner must be integral part of grid computing resource management systems.

  Support for the development of fault-tolerant applications has been identified as one of the major technical challenges to address for the successful deployment of computational grids [24]. To date, there has been limited support for application-level fault tolerance in computational grids. Support has consisted mainly of failure detection services or fault-tolerance capabilities in specialized grid toolkits. Neither solution is satisfactory in the long run. The former places the burden of incorporating fault-tolerance techniques into the hands of application programmers, while the latter only works for specialized applications. Even in cases where fault-tolerance techniques have been integrated into programming tools, these solutions have generally been point solutions, i.e., tool developers have started from scratch in implementing their solution and have not shared, nor reused, any fault-tolerance code. A better way is to use the compositional approach in which

fault-tolerance experts write algorithms and encapsulate them into reusable code artifacts, or modules.

- **Monitoring Systems:** In this a fault monitoring unit is attached with the grid. The base technique which most of the monitoring units follow is heartbeating technique. The heartbeating technique [25] is further classified into 3 types:
    - Centralized Heartbeating - Sending heartbeats to a central member creates a hot spot, an instance of high asymptotic complexity.
    - Ring Based Heartbeating - along a virtual ring suffers from unpredictable failure detection times when there are multiple failures, an instance of the perturbation effect.
    - All-to-all heartbeating - sending heartbeats to all members, causes the message load in the network to grow quadratically with group size, again an instance of high asymptotic complexity
- **Checkpointing-recovery:** Checkpointing and rollback recovery provides an effective technique for tolerating transient resource failures, and for avoiding total loss of results. Checkpointing involves saving enough state information of an executing program on a stable storage so that, if required, the program can be re-executed starting from the state recorded in the checkpoints. Checkpointing distributed applications is more complicated than Checkpointing the ones which are not distributed. When an application is distributed, the Checkpointing algorithm not only has to capture the state of all individual processes, but it also has to capture the state of all the communication channels effectively. Checkpointing [26] is basically divided into 2 types:
    - Uncoordinated Checkpoint: In this approach, each of the processes that are part of the system determines their local checkpoints individually. During restart, these checkpoints have to be searched in order to construct a consistent global checkpoint.
    - Coordinated Checkpoint: In this approach, the Checkpointing is orchestrated such that the set of individual checkpoints always results in a consistent global checkpoint. This minimizes the storage overhead, since only a single global checkpoint needs to be maintained on stable

storage. Algorithms used in this approach are blocking and non-blocking.

- **Fault Tolerant Scheduling:** With the momentum gaining for the grid computing systems, the issue of deploying support for integrated scheduling and fault-tolerant approaches becomes a paramount importance [27]. For this most of the fault tolerant scheduling algorithms are using the coupling of scheduling policies with the job replication schemes such that jobs are efficiently and reliably executed. Scheduling policies are further classified on basis of time sharing and space sharing.

Application dependent mechanisms use the reuse concept to integrate fault tolerance within the application code. Most of the present middleware uses heartbeat based monitoring system to monitor the status of grid. In case of Checkpointing-recovery and fault-tolerant scheduling, they are only able to deal with crash failure semantics for both hardware and software components. Software faults with more malign failure semantics – such as timing or omission ones, which are even more difficult to deal with - are not covered by them.

3.5 SURVEY: FAULT TOLERANCE IN DIFFERENT GRID MIDDLEWARES

3.5.1 FAULT TOLERANCE IN GLOBUS

Globus provides a heartbeat service to monitor running processes to detect faults [28]. The application is notified of the failure and expected to take appropriate recovery action.

Many tools are available in the market to handle faults in globus. The basic entities used by each system are: local monitor and data collector.

- A local monitor is responsible for observing the state of both the computer on which it is located and any monitored processes on that computer. It generates periodic “i-am-alive" messages or heartbeats, summarizing this status information.
- A data collector receives heartbeat messages generated by local monitors and actually identities failed components based on missing heartbeats.

Globus Heartbeat Monitor (HBM), which provides a fault detection service for applications developed with the Globus toolkit. HBM comprises three components:

- A local monitor, responsible for monitoring the computer on which it runs, as well as selected processes on that computer.
- A client registration API, which an application uses to specify the processes to be monitored by the local monitor, and to whom heartbeats are sent.
- A data collector API, which enables an application to be noticied about relevant events concerning monitored processes.

3.5.2 FAULT TOLERANCE IN CONDOR-G

Condor [26] provides system-level checkpoints of distributed applications but it not geared to high-performance parallel programs. Condor-G [30] is built to tolerate four types of failure: crash of the JobManager, crash of the machine that manages the remote resource (the machine that hosts the GateKeeper and JobManager), crash of the machine on which the GridManager is executing (or crash of the GridManager alone), and failures in the network connecting the two machines.

The GridManager detects remote failures by periodically probing the JobManagers of all the jobs it manages. If a JobManager fails to respond, the GridManager then probes the GateKeeper for that machine. If the GateKeeper responds, then the GridManager knows that the individual JobManager crashed. Otherwise, either the whole resource management machine crashed or there is a network failure (the GridManager cannot distinguish these two cases). If only the JobManager crashed, the GridManager attempts to start a new JobManager to resume watching the job. Otherwise, the GridManager waits until it can reestablish contact with the remote machine. When it does, it attempts to reconnect to the JobManager. This can fail for two reasons: the JobManager crashed (because the whole machine crashed), or the JobManager exited normally (because the job completed during a network failure). In either case, the GridManager starts a new JobManager, which will resume watching the job or tell the GridManager that the job has completed.

To protect against local failure, all relevant state for each submitted job is stored persistently in the scheduler's job queue. This persistent information allows the GridManager to recover from a local crash. When restarted, the GridManager reads

the information and reconnects to any of the JobManagers that were running at the time of the crash. If a JobManager fails to respond, the GridManager starts a new JobManager to watch that job.

3.5.3 FAULT TOLERANCE IN LEGION

Legion [28] provides mechanisms to support fault tolerance such as Checkpointing, but the policy is left to the application. A component-based reflexive architecture allows fault tolerance methods to be encapsulated in reusable components that user applications may choose from [32].

Two properties characterize reflective systems: *introspection* and *causal connection*. Introspection allows a computational process to have access to its own internal structures. Causal connection enables the process to modify its behavior directly by modifying its internal data structures—there is a cause-and-effect relationship between changing the values of the data structures and the behavior of the process.

The implementation of the reflexive architecture relies on common basic primitives such as:

- intercepting the message stream
- piggybacking information on the message stream
- acting upon the information contained in the message stream
- saving and restoring state
- detecting failure
- exchanging protocol information between participants of an algorithm

3.5.4 FAULT TOLERANCE IN SUN N1 GRID ENGINE

N1GE6 has several fault tolerance features. The main among them is Checkpointing. Checkpointing in N1GE6 can be classified in 2 different classes [33]:

- Kernel-level
    - Such tools are built into the kernel of the operating system. During a checkpoint, the entire process space (which tends to be huge) is written to physical storage.
    - The user does not need to recompile/re-link their applications.

- Checkpointing and restarting of application is usually done through OS commands.
- Checkpointed application is usually unable to be restarted on a different host.

- User-level
  - These “tools” are built into the application which will periodically write their status information into physical storage.
  - Checkpointing of such applications is usually done by sending a specific signal to the application.
  - Restarting of such applications is usually done by calling the application with additional parameters pointing to the location of restart files.

N1GE6 has built-in support for the integration of 3rd party checkpointing tools. Certain checkpointing tools (mostly user-level) allow the restart of applications on different hosts. These tools, coupled with the migration support on the N1GE6 and with proper configuration of the queue threshold levels, allow the administrator to finely load balance the N1GE6 cluster. One such tool is Berkeley Lab Checkpoint/Restart (BLCR). BLCR is a kernel module that allows you to save a process to a file and restore the process from the file. This file is called a context file. A context file is similar to a core file, but a context file holds enough information to continue running the process. A context file can be created at any point in a process's execution. The process may be resumed from that point at a later time, or even on a different workstation.

3.5.5 FAULT TOLERANCE IN ALCHEMI

Alchemi [34] is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct desktop grids and develop grid applications. Till today not much work is done on Alchemi Fault tolerance. The basic technique used by Alchemi for Fault Tolerance is Heartbeating. The Executors in the Alchemi sends the heartbeat signals to the manager at regular interval of time. Manager after receiving signals from executors guesses that the

executor node is still working. So the procedure of fault tolerance is not so well planned. In later chapters we will see fault tolerance procedure of Alchemi in more detail and try to find challenges in its fault tolerance system and how they can removed.

3.6 THE GREATEST PROBLEMS FOR RECOVERING FROM A FAILURE

The greatest problem is to diagnose the failure, i.e. to identify its root cause [35]. The difficulty to implement the application- dependent failure recovery behavior is next main concern (the user does not know what to do to recover from a failure). To gain authorization to correct the faulty component is another important concern. Other problems such as ensuring that failures do not result in orphaned jobs on remote systems (i.e. they get cleaned up in a reasonable time), cleaning up corrupted cache files without losing lots of work in progress, and getting access to preserved state when Checkpointing/recovery is used (e.g. checkpoint files may be inaccessible or totally lost) were also highlighted.

- Diagnosing the Failure
- Difficulty to implement the failure recovery behavior
- Gain authorization to correct the faulty components

3.7 STANDARDS

Two popular industry standard monitoring systems that are working for implementing fault tolerance in grids are:

3.7.1 GMA

The Grid Monitoring Architecture models [36] the information infrastructure of the grid as a set of Producers that provide information, and a set of Consumers that request information. A Registry manages interaction between Producers and Consumers. Producers contact the Registry to store information about the data they provide. Consumers contact the Registry to find Producers that publish information they require. The Registry returns to the Consumer a list of Producers that provide the desired information and the Consumer then contacts the appropriate Producer or Producers to obtain this information.

3.7.2 ReGS

ReGS [37] is an OGSA-based Meta-OS Grid Service for logging, tracing and monitoring applications in a distributed, heterogeneous computer environment. It provides standard logging interfaces for use by other Grid Services and Applications. It is also capable of virtualizing existing logging systems including zOS logging, NT events, and Unix syslogs. Filtered log messages are managed by ReGS for synchronous and/or asynchronous delivery to consumers. Filtering can occur on different criteria, e.g., message severity or application-specific and filter definitions are extensible. ReGS exploits OGSI interfaces and will be implemented on top of existing messaging systems (e.g., MQSI). ReGS is the plumbing behind more advanced functions such as end-to-end problem determination.

3.8 Algorithms for Fault Tolerance

Some of the Algorithms that are widely used for handling faults in grids are:

3.8.1 Dynamic Heartbeat Grouping Algorithm

The failure detection approach using heartbeat groups within a Grid requires the use of a robust and efficient grouping algorithm that incorporates new members into the failure detection system, relocates members from a group that has too few members and spawns new groups when group membership exceeds the ceiling [25]. It avoids situations where a heartbeat group has very few members while another group in the same sub-cluster has a comparatively large number of members. It also avoids scenarios in which a large number of heartbeat groups are formed, each, with very few members. The main objectives of the algorithm are:

i) Minimize the delay between a membership request by a node and its eventual acceptance into a group. Acceptance of a node into the right group Groups with minimum membership are the most preferred ones and the ones with maximum memberships, are the least preferred ones.

ii) Minimize the number of messages required for the whole exercise.

iii) Keep the overall Grid membership scenario stable for efficient working of the failure detection mechanism.This algorithm considers the two cases:

- The New node(s) join the Grid
- Node(s) failed or detached from the grid.

3.8.2 CONSENSUS ALGORITHM

In the Consensus algorithm, all functioning processes must propose and unanimously agree on a value, in spite of the fact that any of the participating processes may fail during the execution of the algorithm. Consensus has interesting practical uses as it can be used to implement essential fault-tolerant functions such as leader election and voting.

A fundamental result from distributed systems is that Consensus can not be implemented in a distributed system subject to crash failures if the system is asynchronous, i.e., if no timing assumptions can be made, such as the amount of time it takes for two processes to communicate or the amount of time it takes for an operation to complete [38].

One solution to this problem is to augment the distributed system with additional information, such as knowledge about which system components have failed. Given this information, it is possible to implement Consensus in the presence of crash failure. Less formal failure behaviors, such as restart, can also be defined with respect to failure detection. Failure detection provides a sound foundation on which to build a range of failure behaviors and as such, we conclude that it should be provided as a basic service in distributed computing environments.

It is interesting to consider what are the weakest properties that a failure detector can have and still be useful. Consensus in asynchronous distributed systems can be solved with an unreliable failure detector that can erroneously indicate that a component has failed only to correct this error at a later time [39]. Furthermore, an unreliable failure detector can be distributed, with each component of the system having access to its own detector and each detector potentially producing a different account of which system components have failed. This result holds as long as the failure detector meets some minimal requirements for completeness and accuracy. In particular:

- all failed components are eventually discovered and permanently identified as such, and

- at least one functioning component is known to be functioning by all functioning components in the system after some point in time.

3.8.3 CHECKPOINTING ALGORITHM

Checkpointing refers to the operation of saving the execution state of a running computation. In order to save a consistent snapshot of any group of communicating objects, it is necessary not to modify any object while the checkpointing of the group is in progress. This can be ensured in two ways: blocking and unblocking. In the blocking method, all threads suspend themselves during the first part. In the non-blocking method, all threads execute in this phase, but they are responsible for not modifying any objects that are not checkpointed yet [40]. It was seen during the evaluation of the schemes that the overhead due to blocking is not significantly high, and therefore, the additional complexity of the non-blocking schemes is unnecessary. The threads are blocked only for the period in which object checkpoints are in progress. The period in which threads save their states is not required to be blocking, and execution of some threads may continue while others are saving (or yet to save) their state. For similar reasons, rollback procedures could also either be blocking or non-blocking.

In order to create a consistent global checkpoint, all components are first frozen, the individual checkpoints are taken, and all the components are then un-frozen [41]. The distributed checkpointing algorithm is implemented in parallel - new threads are created for every component that is part of the application, and the checkpoints are stored with the Storage services by the components independently. Hence, in theory, increasing the number of components should not affect the performance of the application as long the checkpoint sizes remain the same. However, in practice, there may be a few minor overheads for every additional component, viz. thread creation, extra handle resolutions, etc.

The above statement is true as long as the Storage services scale well with the increasing number of components. In addition, the performance of the algorithm also depends on the size of the checkpoints. In specific, the checkpoint time would depend on the time taken by the component with the largest checkpoint size to store its checkpoint, and is expected to be linear with respect to it.

3.9 CONCLUSION

This chapter described the related research done in the area of fault tolerance in grid computing. We started with the introduction of computational grids, their characteristics and the factors that are to be considered while constructing the computational grids. Various kinds of faults that can normally occur in grids have also been defined and the approaches to handle these faults like Checkpointing, replication, application dependent and scheduling based have been described in brief. As the size of grid increases the chances of fault occurrence increases, so to deal with it on large scale, standard monitoring systems like GMA and ReGS have been introduced under guidance of standard authorities. One can also use some algorithms for creating a fault tolerant system for grids. Few of such algorithms have also been discussed in this chapter.

The next chapter will discuss the research findings after studying the computational grids.

RESEARCH FINDINGS 4

Conventional resource management schemes are based on relatively static model and a centralized controller that manages jobs and resources accordingly. However, this fails to work if the centralized controller fails due to many reasons discussed in chapter3. Secondly there are issues like what will happen if some job hangs down or remain unfinished in between its execution? To overcome these issues, we need an adaptive infrastructure which is robust, flexible and adaptive in nature. This part of the thesis will discuss fault tolerance paradigm that focuses on dealing with such kind of failures. The proposed framework can manage the central manager failure and also tries to keep a check on the correct completion of the job in Alchemi based computational grids.

This chapter is organized as follows: The section 4.1 discusses about the general Alchemi based Computational Grids; section 4.2 discusses the selection of Alchemi as framework for establishing the Computational Grid Environment. The section 4.3 of this chapter discusses the Fault tolerance in Alchemi. Section 4.4 concludes this chapter.

4.1 ALCHEMI BASED COMPUTATIONAL GRIDS

Alchemi is an open source software framework that allows you to painlessly aggregate the computing power of networked machines into a virtual supercomputer (**computational grid**) and to develop applications to run on the grid.

Alchemi includes:

- The runtime machinery (Windows executables) to construct computational grids.
- A .NET API and tools to develop .NET grid applications and grid-enabled legacy applications.

It has been designed with the primary goal of being easy to use without sacrificing power and flexibility. As Alchemi is the emerging technology, so a lot of work has to be done to make it a standard .NET based middleware. We are using Alchemi as the framework to setup the computational grids.

4.2 WHY ALCHEMI?

There can be many reasons for choosing Alchemi as the framework for building the computational grid and then further choosing to build a fault tolerant system for it. Some of these reasons are:

- Alchemi is the first .NET based stable grid.
- Most important is that Alchemi is Open Source. So one can do any number of changes in it.
- Another important reason is that most of the systems in our labs are running Windows Operating Systems.
- Fault Tolerance research area is still under development in Alchemi.

4.3 FAULT TOLERANCE IN ALCHEMI

Fault tolerance is the property of a system that helps it to continue operating consistently with its specifications even in the event of failure of some of its parts. From a user's point of view, a distributed application should continue despite failures. Alchemi based distributed system is characterized by a collection of autonomous processing elements, called nodes. Each of these nodes has some computing resources as well as the possibility to exchange information with some of the other nodes. These are referred to as its neighbors and the communication between these nodes take place through a central authority called manager. For more detail on Alchemi refer to Appendix A.

The central manager controls the working of all the executors (Figure 4.1). Users interact with the central manager for submitting and enquiring the status of their jobs. 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-enabled 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 them to the Manager for execution by the grid.

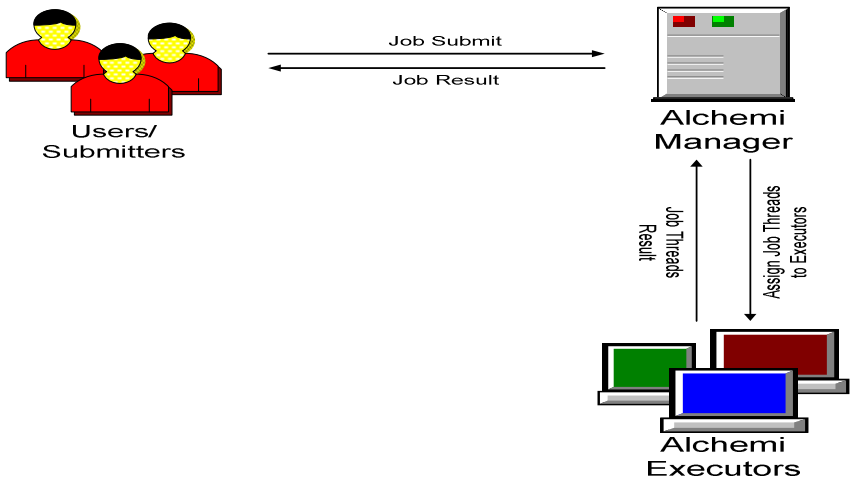


Figure 4.1: Sample Alchemi Computational Grid

4.3.1 PRESENT FAULT TOLERANCE SCENARIOS IN ALCHEMI

1. **Heartbeat mechanism is used by Alchemi:** Executors send the heartbeat messages at some interval to the manager to whom they are connected. This will help the manager to maintain the status of the executors. In case the manager doesn't receive the heartbeat messages from the executor between the pre-decided times, the manager consider that executor to be dead and update its information. Here the reason for not receiving of the message is considered to be hardware failure, OS reboot or process being killed. These are termed as hard failure of executor.
2. **Executor fails "hard"** due to a hardware failure, an OS reboot or the process being killed. In this case Alchemi is using the heartbeat technique to re-schedule the thread to another Executor.
3. **Executor fails "soft"** due to the user logging off or stopping down the Executor. In this case the Manager is informed that the Executor is going offline and the thread is re-scheduled (this scenario fails many times). This is an acceptable solution.
4. **Restart the incomplete job:** In case if the job remains unfinished due to any reason like- hardware failure, failure of the manager, etc the alchemi manager stores the status of the job by using "applicatio_id", "internal_thread_id" and "executor_id" fields it maintain in the SQL database.
5. **Dynamically determine new nodes added or deleted:** As the new node is added or removed, the manager is being provided with the information dynamically and it updates the console information to make a correct record of the information so as to avoid any faults.

4.3.2 PROBLEMS ASSOCIATED WITH FAULT TOLERANCE IN ALCHEMI

1. **System fails if the Manager node fails:** In case of a Manager failure there is no backup at this point. An immediate problem here is that once a Manager goes offline all Executors that are running threads will probably fail as well.

2. **Executing thread "hangs"** on the Executor. This means that the executing thread is not responding but the heartbeat thread keeps working. Currently the Executor remains hanged until it is restarted. This is not an acceptable solution.
3. **Many times executor remains connected state even when disconnected.**

4.4 QUESTIONS ANSWERED BY THIS THESIS

- *What are the more frequent kinds of failures one can face on Alchemi Grids?*
  To identify the kind of faults that can occur in Alchemi based grid environment. Faults due to Configuration, Middleware, Application and Hardware may occur in it.
- *What is current fault tolerance scenarios used in Alchemi for Fault Tolerance?*
  The present fault tolerance scenarios are found and discussed in the section 4.3.1. Of these the scenario 2 can be further improved by using the Checkpointing technique instead of log based technique.
- *What are the deficiencies in the fault tolerance system used by Alchemi?*
  Some of the deficiencies identified are described in section 4.3.2. This thesis will try to deal with these kinds of failures.
- *How to handle these deficiencies?*
  For the case in which the central manager fails, we are going to provide the backup manager that will use the kind of heartbeat technique. The backup manager will come into play when the Central Manager fails. For dealing with the problem of executor thread hangs, we are proposing the TTL concept.

4.5 CONCLUSION

This chapter discussed Alchemi as the .NET based framework. The reasons for selection of Alchemi as our base framework for implementing our fault tolerance system have also been defined. After selecting the particular framework, in our case Alchemi, we tried to find out what are the present mechanisms for handling faults and what are the deficiencies still present there in Alchemi framework for handling the faults.

In next chapter we will study the implementation and experimentation of proposed work.

PROPOSED TECHNIQUES 5

Implementation is the process of realizing a project in practice according to the proposed work. Till now we have gathered lot of information regarding the fault tolerance in Computational grids. This Chapter focuses on the implementation details. A frame work for the efficient fault tolerance in the Alchemi based Computational grids is defined. This proposed framework tries to answer all the questions described in previous chapter. For implementing it we have to first setup the Grid environment using Alchemi. After setting up the grid environment, we have implemented our fault tolerance system.

This chapter first discusses the basic requirements that are needed for implementation of our project. Section 5.2 covers the details regarding the implementation that includes grid setup and proposing the framework for our fault tolerance system. At end of this section, we provide the implementation steps. After the implementation is over, in section 5.3 we evaluated proposed system by considering various test cases.

5.1 IMPLEMENTATION REQUIREMENTS

The required configuration for the implementation of my proposed work is

5.1.1 HARDWARE REQUIREMENTS

The minimum configuration for hardware is:

1. Personal computers – PIII, PIV
2. Local Area Networks

5.1.2 SOFTWARE REQUIREMENTS

The minimum configuration for software is:

1. Microsoft Windows 9x/NT/2000/XP/2003
2. Microsoft .Net framework 1.1
3. Microsoft SQL Server 2000 or MSDE 2000 installed with security
4. ASP.NET
5. Internet Information Services (IIS)
6. Alchemi 1.03
7. Visual Studio .net development kit

5.1.3 PROGRAMMING LANGUAGE

Our choice of programming language depended on several factors:

- Platform Independence,
- Object Serialization Support,
- Multithreaded Program Support, and
- GUI support, with consistent user experience (look and feel) across different platforms.

As C# satisfies all these factors, it is the best match for implementing the proposed work in this thesis.

5.2 IMPLEMENTATION DETAIL

In this section we are describing the implementation details. For developing a fault tolerance system, we need to first setup the grid environment. We are going to propose the fault tolerance system and try to implement the backup manager from it.

5.2.1 SETTING UP COMPUTATIONAL GRID: ALCHEMI

We can define a "computational grid" as being any distributed computational network, enabled with software that allows cooperation and the sharing of resources. Computational grids can be developed using many middleware. We are here using the Alchemi: A .NET based middleware for establishing the computational grid environment. The steps necessary to realize a computational grid include:

- The integration of individual software and hardware components into a combined networked resource.
- The implementation of middleware to provide a transparent view of the resources available.
- The development of tools that allows management and control of grid applications and infrastructure.
- The development and optimization of distributed applications to take advantage of the resources.

5.2.1.1 STEPS FOR INSTALLING COMPUTATIONAL GRID: ALCHEMI

These are the steps for installing various Alchemi Components. For every component we are giving its installation, configuration and operations are defined.

1. Alchemi Manager

Installation

The Alchemi Manager can be installed in two modes
- As a normal Windows desktop application
- As a windows service. (supported only on Windows NT/2000/XP/2003)

- To install the manager as a windows application, use the Manager Setup installer. For service mode installation use the Manager Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the “Alchemi Manager Service” installed and configured to run automatically on Windows start-up. After installation, the standard Windows service control manager can be used to control the service. Alternatively the Alchemi ManagerServiceController program can be used. The Manager Service

controller is a graphical interface, which is exactly similar to the normal Manager application.

- Install the Manager via the Manager installer. Use the sa password noted previously to install the database during the installation.

Configuration & Operation

- The Manager can be run from the desktop or Start -> Programs -> Alchemi -> Manager ->
- Alchemi Manager. The database configuration settings used during installation automatically appear when the Manager is first started.
- Click the "Start" button to start the Manager.
- When closed, the Manager is minimised to the system tray.

2. Cross Platform Manager

Installation

- Install the XPManager web service via the Cross Platform Manager installer.

Configuration

- If the XPManager is installed on a different machine that the Manager, or if the default port of the Manager is changed, the web service's configuration must be modified. The XPManager is configured via the ASP.NET Web.config file located in the installation directory (wwwroot\Alchemi\CrossPlatformManager by default):

```
<appSettings>
<add key="ManagerUri" value="tcp://localhost:9000/Alchemi_Node" />
</appSettings>
```

Operation

- The XPManager web service URL is of the format

http://[host_name]/[installation_path]

- The default is therefore

http://[host_name]/Alchemi/CrossPlatformManager

- The web service interfaces with the Manager. The Manager must therefore be running and started for the web service to work.

3. Executor

Installation

The Alchemi Executor can be installed in two modes

- As a normal Windows desktop application
- As a windows service. (Supported only on Windows NT/2000/XP/2003)

- To install the executor as a windows application, use the Executor Setup installer. For servicemode installation use the Executor Service Setup. The configuration steps are the same for both modes. In case of the service-mode, the "Alchemi Executor Service" installed and configured to run automatically on Windows start-up. After installation, the standard Windows service control manager can be used to control the service. Alternatively the Alchemi ExecutorServiceController program can be used. The Executor service controller is a graphical interface, which looks very similar to the normal Executor application.
- Install the Executor via the Executor installer and follow the on-screen instructions.

Configuration & Operation

The Executor can be run from the desktop or Start -> Programs -> Alchemi -> Executor -> Alchemi Executor.

The Executor is configured from the application itself.

You need to configure 2 aspects of the Executor:

- The host and port of the Manager to connect to.
- Dedicated / non-dedicated execution. A non-dedicated Executor executes grid threads on a voluntary basis (it requests threads to execute from the Manager), while a dedicated Executor is always executing grid threads (it is directly

provided grid threads to execute by the Manager). A non-dedicated Executor works behind firewalls.

- Click the "Connect" button to connect the Executor to the Manager.

5.2.2 PROPOSING FRAMEWORK

In the proposed work we are giving different ways to monitor the faults in Alchemi based computational grid. The monitoring system in Alchemi consists of Managing Unit called Manager, to which all the execution nodes called 'executors' which report their status. Figure 5.1 shows the interaction between the user, manager and the executors. The execution units keep on sending the heartbeat signals at regular intervals to the manager telling about their current status. The heartbeat gives two kind of information to the Manager.

- Job Status: The executor sends the message to the Manager if a job process executing on it exists. The message specifies if the process exited gracefully or due to some fault. If gracefully exited then the manager simply mark the job as finished and display the results to the manager. Also the manager deletes all the threads related to this job from its database. If due to some reason if the thread exits without completion of the job, the manager save the current state of the thread and restart the job from the current state at some different resource. For maintaining the current state, techniques like Checkpointing or logs can be used.
- Machine Status: The monitoring units at the executors keep on sending the regular heart beat messages to the manager. Through this message manager knows that the execution node is alive and is working. If the manager doesn't receive the heart beat signal from some executor, the manager simply declares it as dead and updates its information.

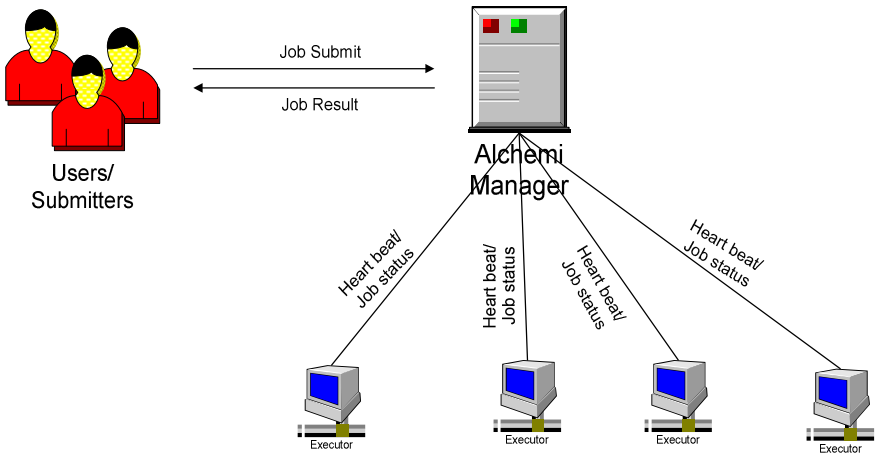


Figure 5.1: Monitoring Architecture

5.2.2.1 PROPOSING THE BACKUP MANAGER

While working with the setup grid, the failure of the centralized manager causes the whole grid to hangs down. Till the manager doesn't restart, the grid remains inaccessible. So to avoid such kind of grid failure, we have introduced the concept of the **backup manager**. The backup manager also uses the heart beat phenomenon to query the status of manager. The steps for implementing the backup manager are:

- After every heartbeat interval, the leader node sends a packet to the backup manager. Figure 2 shows the Backup manager concept.
- Packet received by the backup manager contains information about each of the nodes in the group and its arrival indicates that the leader is up and running.
- The backup leader updates its database using the data obtained from the received packet.
- In case of absence of packet for certain predefined time interval lets the backup manager to consider the master has failed, and it itself takes the control as new master.
- This change is multicast to the executors and they update their connections database in order to accept Backup Manager as their new master. Figure 3 shows the new connection established between the executors and the backup manager.

We try to use these steps for providing the support for backup manager for Alchemi based computational grids. The concept of SQL replication is used for creating the replication server. From there the backup manager accesses the database to control the grid.

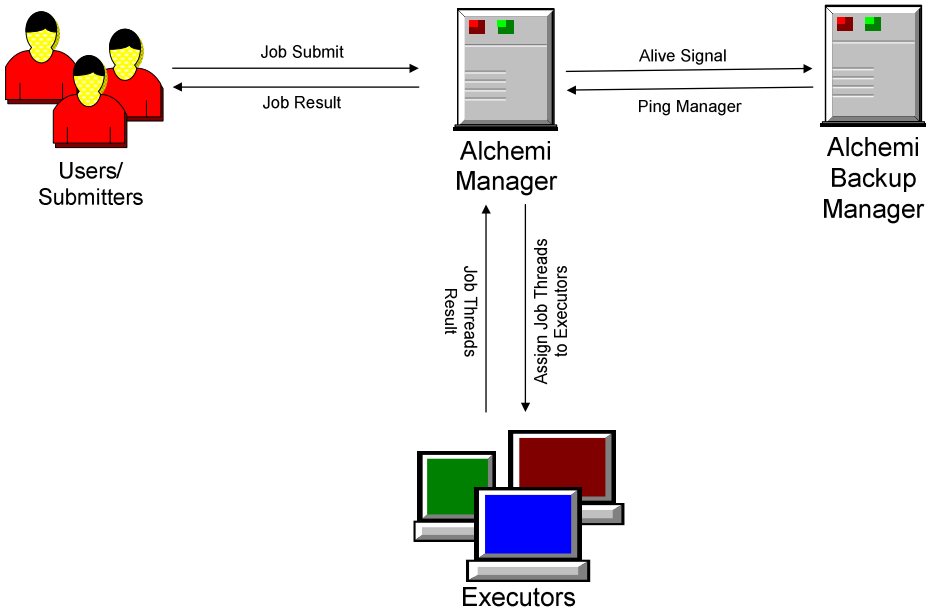


Figure 5.2: Backup Manager Concept

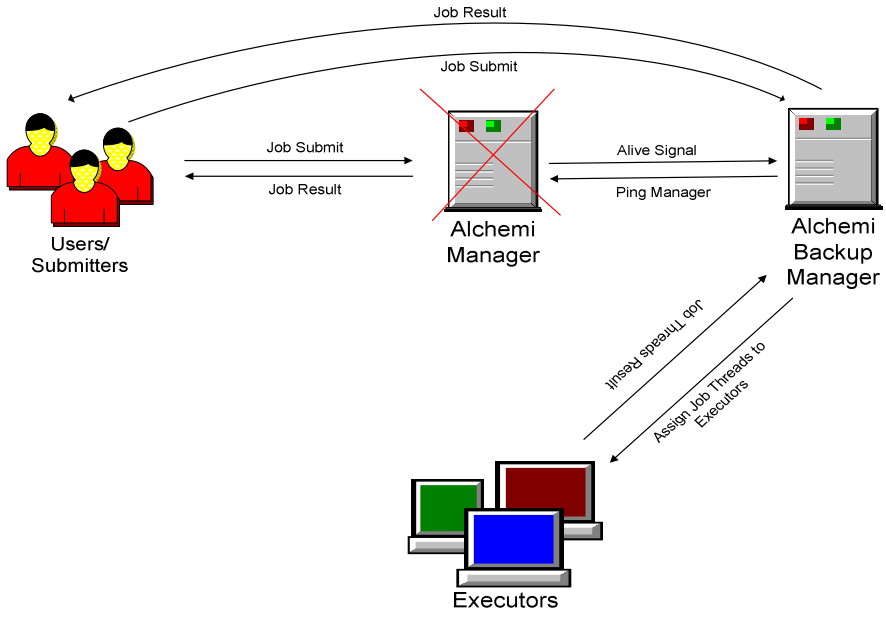


Figure 5.3: Failure of Manager Leads Backup Manager to take the control

5.2.3 IMPLEMENTATION OF BACKUP MANAGER

Backup Manager is implemented in following three steps:

Step1: Creating the Port Scanner

We need port scanner kind of application to check for the status of the manager. In our port scanner application, application tends to scan port 9000 of the remote system where manager is running at regular interval of time.

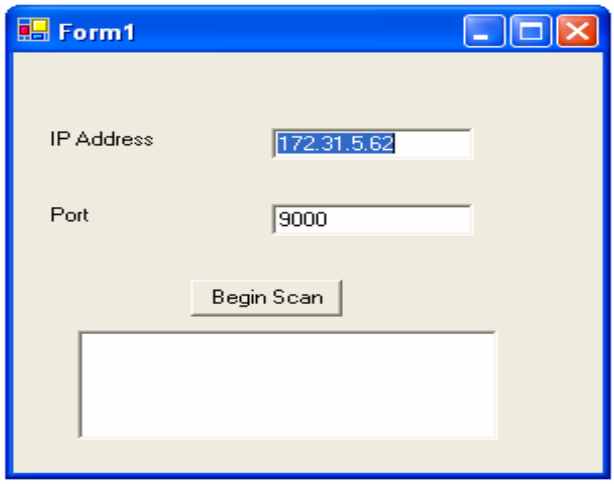


Figure 5.4: Port Scanner

The code for this application is written in c#. The two main classes used for this are: 'Socket' and 'IPEndPoint'. User has to provide the IP address of the system where manager is running. In case when the manager fails due to any reason, a message showing manager failure is displayed at Backup Manager. Now the backup manager starts acting as the new central manager.

Step2: Creating the Replication Database

Replication is the process of sharing data between databases in different locations. Using replication, we can create copies of the database and share the copy with different users so that they can make changes to their local copy of database and later synchronize the changes to the source database. For detail regarding database to be replicated refer to Appendix C.

1. Open SQL Server Enterprise Manager and select Tools menu -> Replication -> Configure Publishing, Subscribers, and Distribution…
    a. Configure the appropriate server as publisher or distributor.
    b. Enable the appropriate database for merge replication.
    c. Enable the appropriate server as subscriber.
2. Open SQL Server Enterprise Manager and select the appropriate SQL Server Group for which replication needs to be done, then select Tools menu -> Replication -> Create and Manage Publications.
3. This will open a dialog box for "Create and Manage Publications on respective server". Select the appropriate database and then click "Create Publication". This will open "Create Publication Wizard". Just click *Next*.
4. It will ask to choose a Distributor for the selected server. Select "Make Server its own Distributor; SQL Server will create a distribution database and a log". Then click *Next*.
5. It will ask for the Snapshot folder path. Browse and select the appropriate path for Snapshot folder and then click *Next*.
6. Choose the Alchemi database to be published and Click Next.
7. Select the Publication Type as "Merge Publication".
8. Specify the Subscriber Types. Select "Servers running SQL Server 2000". Then click *Next*.
9. Select the Object Types (name of tables present in the Alchemi database) which you want to publish, and click *Next*.
10. It will show some issues which may require some changes at later stages in order to work as expected. Just click *Next*.
11. Give Publication Name and click *Next*.
12. It will ask to customize the properties of the Publication. Select "No" and go by default seetings.
13. It will show "Set Snapshot Agent Schedule" dialog box. Change the Snapshot Agent Schedule as per your requirement, then select "Create the first snapshot immediately". And click *Next*.
14. Click Finish to create a Publication.
15. Finally, it will show "SQL Server Enterprise Manager successfully created publication 'pub1' from database 'Alchemi'. Just click *Close*.

16. It will show the dialog box “Create and Manage Publications on respective Server”. Now go to the Alchemi1 Publication and click “Push New Subscription”.
17. Before doing “Push New Subscription”, create new SQL Server Registration for Subscriber machine in the Publisher machine’s SQL Server Enterprise manager with SQL Authentication mode. For this, there should be one common SQL login name in both Publisher and Subscriber machines.
18. Go to “Push New Subscription” wizard. This will open “Push Subscription Wizard”. Just Click Next.
19. Choose one or more subscribers from Enabled Subscribers and click *Next*.
20. Choose Subscription (destination) database name by browsing and clicking *Next*.
21. “Set Merge Agent Schedule”. Change the Schedule as per your requirement and click *Next*.
22. Specify whether the Subscription(s) needs to be initialized or not. Select “Yes, initialize the schema and data” as well as select “Start the Snapshot Agent to begin the initialization process immediately”, and click *Next*.
23. “Set Subscription Priority” as “Use the Publisher as a proxy for the Subscriber when resolving conflicts”, and click *Next*.
24. It will show the status of the SQLSERVERAGENT service as running. Just click *Next*.
25. Click *Finish* to complete the Push Subscription.
26. Finally, it will show “Subscriptions were created successfully at the following Subscribers:”. Just click *Close*.
27. Now, in the SQL Server Enterprise Manager, go to the appropriate SQL Server Group and go to “Replication Monitor -> Publishers -> Respective Server -> Publication Name”. In the right pane, you will see the snapshot agent. Just right click and select “Start Agent”. Refresh it once. Then right click on the respective publication name and select “Start Synchronizing”. It will merge the necessary data. Refresh it once.

Step3: Creating Multicasting Application

IP Multicasting is a bandwidth-conserving technology that reduces traffic by simultaneously delivering a single stream of information to thousands of corporate

recipients and homes. A few things must be in place before developing multicasting applications or migrating unicast applications.

Network Requirements

For IP multicasting to work all the routers along the path of communication must be multicast enabled.

System Requirements

The operating system networking interface must support multicasting.

5.3 EXPERIMENTAL EVALUATION

After successfully building the backup manager, we had tested it in three stages. We first form the test strategy, and then define the test cases and finally we present the results.

5.3.1 TEST STRATEGY

In our implementation, we have taken a part of total 24 nodes at two different departments of the TIET Deemed University in our computational grids. For building the application for our system, we have used C# as programming language and MS SQL Server as the database. The backup manager for managing the proper working of manager is implemented using the steps described in section 5.2.2.1. We have tested our system by running the sample application called PiCalculator.

5.3.2 TEST CASES

Following are the test cases that we are going to use to check our system.

Test Case #1 Executor On/Off

We dynamically switch on and off the executors. While doing this we kept on checking the status of the executing jobs through the Alchemi Console Manager.

Test Case #2 Heartbeating between Manager and Backup Manager

We have designed the small module called “Port Scanner” that kept on checking the central manager for its working. This module is just like a heartbeat monitoring system, in which the backup manager invokes the central manager to give a heart beat signal at a regular interval of time. To test this we have noted the results of port scanner after specified interval of time.

Test Case #3 Automatic Replication

We check this case with the help of replication monitor facility provided in the MS SQL server. It uses the snapshot agents to update the database and present information.

Test Case #4 Manager Stopped

This case is tested in two parts. First, by manually disconnecting the manager and note the results and secondly, by switching off the system without disconnecting the manager.

5.3.3 EXPERIMENTAL RESULTS

While testing we check many possibility of faults occurrence. We have checked it for four cases discussed above. We found that our proposed system helps to increase the performance of grid by providing the backup manager concept. When the central manger fails the backup manager successfully comes into play. In test case 1, while executing the job Pi Calculator, failure of some executor still let the result for application to be displayed. Only the difference was in the time taken by job to be finished. We first tried with four executors (Figure 5.5), application took 00:01:01.0265500 and if we stop one executor in between, the job was finished in 00:01:09.0937500. For test case two, we noted the result of the port scanner (Figure 5.4) at an interval of 10sec; the scanner was successfully monitoring the central manager. The test case three was tested by using the Microsoft SQL Server built in facility to monitor replica. Test case 4 is the main test case that was to be tested. In this we have defined the two sub cases. When both theses are applied, the backup manager successfully comes into play. Executors have to be reconnected to this backup manager.

5.4 CONCLUSION

This chapter deals with the implementation part of thesis work. After giving the basic requirements for our implementation we have proposed the system that deals with the problems found in Alchemi based computational grids. The system is implemented using C# as programming language and MS SQL Server as Database. The experimental evaluation for our implemented system was also been done by considering different test cases.

In the next chapter we will conclude our thesis work and gave future scope of it.

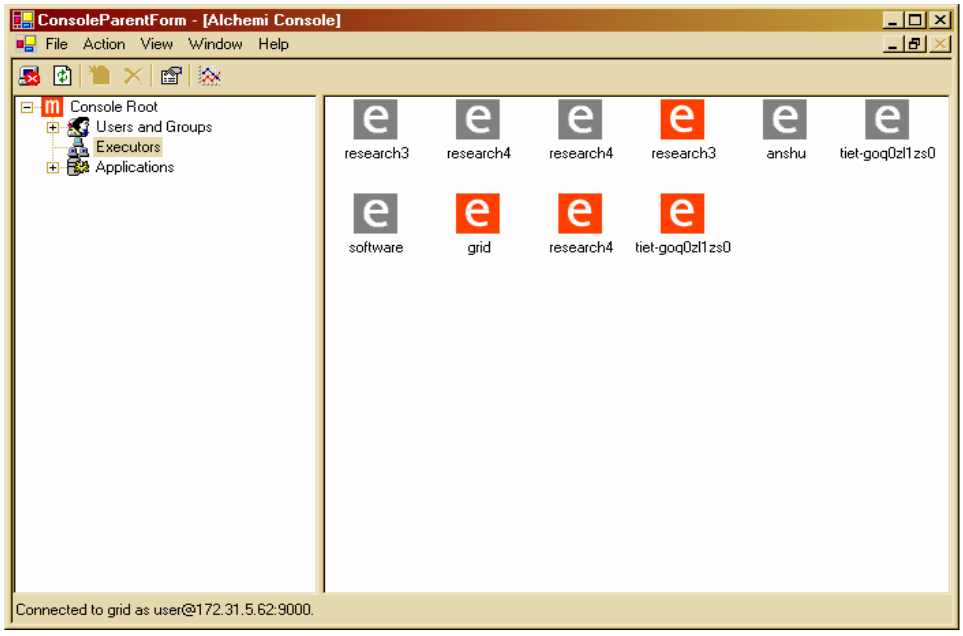


Figure 5.5: Four Executors running while executing Pi Calculator

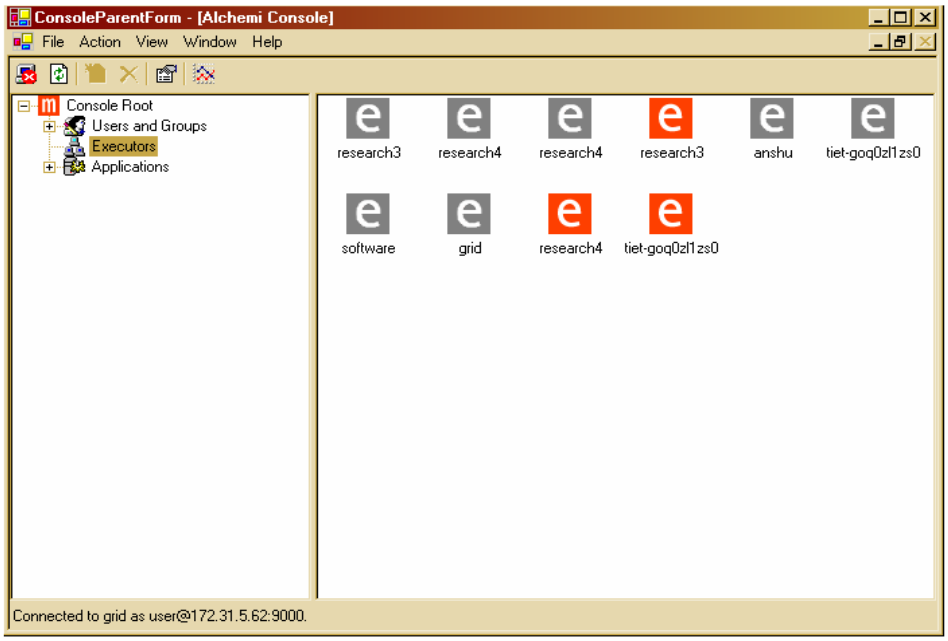


Figure 5.6: One Executor stops while application running

CONCLUSION AND FUTURE WORK 6

The sharing of computational resources is a main motivation for constructing Computational Grids in which multiple computers are connected by a communication network. Due to the instability of grids, the fault detection and fault recovery is a very critical task that must be addressed. The need for fault tolerance increases as the number of processors and the duration of computation increases.
In this thesis, we investigated the issues and challenges involved in dealing with faults in computational grids, proposed the framework for dealing with various kinds of faults we found in Alchemi based computational grids and evaluated the efficiency of our proposed system under various conditions.

This chapter provides some concluding remarks, highlights the main contribution of this thesis and presents some possible future research directions. This chapter is organized as follows: Section 6.1 summarizes the project, section 6.2 presents the main contribution of this thesis, and finally section 6.3 highlights the several possible future research directions.

6.1 SUMMARY

Advances in networking technology and computational infrastructure make it possible to construct large-scale high performance distributed computing environments that provide dependable, consistent and pervasive access to high end computational and heterogeneous resources despite geographical distribution of both resources and users. In a heterogeneous computing environment like Grid, a suite of different machines ranging from personal computer to supercomputer is loosely inter-connected to provide a variety of computational capabilities to execute collections of application tasks that have diverse requirements. These kinds of large scale high performance distributed services have recently received substantial interest from both research as well as industrial point of view. However, an important research problem for such systems is the lack of fault tolerance system. The heterogeneous and dynamic nature of grids makes them more prone to failures than traditional computational platforms. So, the system managing such infrastructures needs to be smart and efficient to overcome the challenge of fault detection and recovery.

Many kinds of failures can occur in computational grids. Different ways to handle these failures are discussed in the literature review section. After studying fault tolerance in different grid middlewares, we have chosen Alchemi based Computational grids the reasons for which have been given in chapter 4. After analyzing the detailed performance of Alchemi, we have identified different areas related to fault tolerance in which this middleware still lacks. It has been realized that, in order to support fault tolerance in computational grid (Alchemi Framework based), it is useful to provide a backup manager support for avoiding the grid to be down when central manager fails.

6.2 MAIN CONTRIBUTIONS

- We have studied the Fault Tolerance in different grid middlewares and found most common kind of failures that can let the computational grid to work improperly. Different fault tolerance techniques are also been discussed in this thesis.

- We have setup the computational grid for our campus and run and develop different applications for studying its proper performance and working. In the setup grid we have include 24 nodes from two different departments.
- Various existing fault tolerance mechanisms already present in the setup grid are identified. After studying the internals of Alchemi we have identified different deficiencies that are their in Alchemi for handling the faults dynamically.
- On the basis of identified deficiencies, we have proposed and implemented backup manager concept that will help in avoiding the grid to become inaccessible if the central manager fails.
- We have experimented and tested our implemented system. It will help in making the grid to work correctly even after the failure of central manager. The limitation of our proposed method is that we have to do all work manually.

6.3 FUTURE WORK

The future extension of the project can be done in three ways:

- Extension of the thesis implementation:
    - The current implementation is done within LAN with limited number of PCs, this can be extended for systems outside the network.
    - Our implementation is manual. If the central manager fails, then the executors are told to connect to the new manager. But this is to be done manually. There must be some provision so that on failure of central manager executors will automatically connect to new manager.
    - This implementation is kind of third party software that will improve the grid fault tolerance. But this can be integrated into the Alchemi middleware code to help the user to easily use it.

- Extension of proposed method:
    - Our approach of backup manager can be extended to work on support for multiclustering with peer-to-peer communication between Managers.
    - Handle execution thread hanging problem must be fixed. Our two ideas to handle this problem are: One idea is to monitor the executing thread and terminate it if it exceeds a configurable amount of time. This value should be configurable from the application so the user can set it but an override at the Executor level is probably desirable as well. One problem here is that some machines take longer to execute something than others so maybe if the time it waits is a factor of the computer's speed it might be useful. Another idea I've been toying with is to require long running threads to raise status events containing a "percent done" value. The monitoring thread would then have data to see if the thread is dead or just taking longer to complete but still alive. The events could have enough information to be used as a checkpoints but this would be up to the implementation of each application.
- Extension to improve the performance:
    - To make the status of application threads alchemi is presently using the log based techniques. We can use the Checkpointing and recovery techniques to further enhance the performance of the system.

REFERENCES

[1] Fran Berman, Geoffrey Fox, and Tony Hey, "The Grid: past, present, future", Copyright 2003 John Wiley & Sons, Ltd.

[2] David De Roure, Mark A. Baker, Nicholas R. Jennings and Nigel R. Shadbolt, " The Evolution of the Grid "pages 65--100. John Wiley and Sons Ltd, 2003.

[3] http://www.gridcomputing.com

[4] GGF, www.ggf.org

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

[6] http://gridcafe.web.cern.ch/gridcafe/

[7] Foster, C.Kesselman, S.Tuecke,"The Anatomy of the GridEnabling Scalable Virtual organizations", 2004 www.globus.org/research/papers/anatomy.pdf

[8] J. Joseph, C. Fellenstein, "Grid Computing", Prentice Hall/IBM Press, Edition 2004

[9] http://www.gridcomputingplanet.com/

[10] en.wikipedia.org/wiki/Grid_computing

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

[12] Frederico Buchholz Maciel, " Resource Management in OGSA", March 1, 2005 http://forge.gridforum.org/projects/cmm-wg/

[13] Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing" Copyright 2001 John Wiley & Sons, Ltd. 17 September 2001.

[14] Kamana Sigdel, "Resource Allocation in Heterogeneous and Dynamic Networks" MS Thesis, Delft University of Technology,2005

[15] Arshad Ali, Ashiq Anjum, Atif Mehmood, et.al. "A Taxonomy and Survey of Grid Resource Planning and Reservation Systems for Grid Enabled Analysis Environment", 2003

[16] GRMS, http://www.gridworkflow.org/snips/gridworkflow/space/GRMS

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

[18] Foster, I., Kesselman, C., “The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann”, pp. 15-51, 1999.

[19] Mark Baker, Rajkumar Buyya and Domenico Laforenza, “The Grid: International Efforts in Global Computing” International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, Rome, Italy, 31 July--6 August 2000.

[20] Paul Townend and Jei Xu, “Fault Tolerance within a Grid Environment” Proceedings of AHM2003

[21] Varun Sekhri, “CompuP2P: A light-weight architecture for Internet computing” MS Thesis, Iowa State University, Iowa, 2005

[22] Yuuki Horita, Kenjiro Taura, Takashi Chikayama, “A Scalable and Efficient Self-Organizing Failure Detector for Grid Applications” 6th IEEE/ACM International Workshop on Grid Computing, Grid 2005

[23] Ya-Fing Zhung, Jizhou Sun, Jian-Bo Ma, “Self Management Model Based on multiagent and Worm Techniques” CCECE 2004

[24] Anh Nguyen-Tuong, “Integrating Fault-Tolerance Techniques in Grid Applications” PhD Thesis, University of Virginia, August 2000

[25] Amit Jain and R.K. Shyamasundar, “Failure Detection and Membership Management in Grid Environments” Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04) pp. 44-52

[26] Sriram Krishnan, “An Architecture for Checkpointing and Migration of Distributed Components on the Grid” PhD Thesis, Department of Computer Science, Indiana University, November 2004

[27] J.H. Abawajy, “Fault-Tolerant Scheduling Policy for Grid Computing Systems”, IPDPS’04

[28] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski ,” A Fault Detection Service for Wide Area Distributed Computations” Volume 2, Number 2, springer, pp. 117-128(12), 1999

[29] M.J. Litzkow et al, “Condor - a hunter of idle workstations,” In Proceedings of the 8th International Conference on Distributed Computing Systems, June 1988.

[30] James Frey, Todd Tannenbaum, Miron Livny Ian Foster, Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids" 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)

[31] A.S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," Communications of the ACM, Vol. 40(1), 1997.

[32] A. Nguyen-Tuong, and A.S. Grimshaw, "Using Reflection to Incorporate Fault-Tolerance Techniques in Distributed Applications," Computer Science Technical Report, University of Virginia, CS 98-34, 1998.

[33] Liang PENG, Lip Kian NG, "N1GE6 Checkpointing and Berkeley Lab Checkpoint/Restart" Dec 28, 2004

[34] Krishna Nadiminti, Akshay Luther, Rajkumar Buyya, "Alchemi: A .NET-based Enterprise Grid System and Framework" December 2005

[35] Raissa Medeiros, Walfredo Cirne, Francisco Brasileiro, Jacques Sauvé, "Faults in Grids: Why are they so bad and what can be done about it?" GRID'03

[36] Rob Byrom, Brian Coghlan, et.al., "Fault tolerance in the R-GMA Information and Monitoring System" EGC2005

[37] http://www.haifa.ibm.com/projects/systems/gc/index.html

[38] M. J. Fischer, N. A. Lynch, and M. S. Paterson., "Impossibility of distributed consensus with one faulty process". Journal of the ACM, 32(2), Apr. 1982.

[39] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems". Journal of the ACM, 43(2), Mar. 1996.

[40] Mangesh Kasbekar Chandramouli Narayanan Chita R Das, "Selective Checkpointing and Rollbacks in Multithreaded Object-oriented Environment" 6th IEEE International On-Line Testing Workshop (IOLTW) p. 3

[41] Sriram Krishnan, "An Architecture for Checkpointing and Migration of Distributed Components on the Grid" PhD Thesis, Department of Computer Science, Indiana University, November 2004

ALCHEMI A

A.1 INTRODUCTION

Alchemi is a .NET-based grid computing framework that provides the runtime machinery and programming environment required to construct desktop grids and develop grid applications. It allows flexible application composition by supporting an object-oriented grid application programming model in addition to a grid job model. Cross-platform support is provided via a web services interface and a flexible execution model that supports dedicated and non-dedicated (voluntary) execution by grid nodes.

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

- Internet-based clustering of Windows-based desktop computers;
- dedicated or non-dedicated (voluntary) execution by individual nodes;
- object-oriented grid application programming model (fine-grained abstraction);
- file-based grid job model (coarse-grained abstraction) for grid-enabling legacy applications and
- web services interface supporting the job model for interoperability with custom grid middleware e.g. for creating a global, cross-platform grid environment via a custom resource broker component.

A.2 DISTRIBUTED COMPONENTS

Four types of nodes (or hosts) take part in desktop grid construction and application execution (see Figure A.3). Deploying a Manager node and deploying one or more Executor nodes configured to connect to the Manager construct an Alchemi desktop grid. 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.2.1 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 Serve (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.

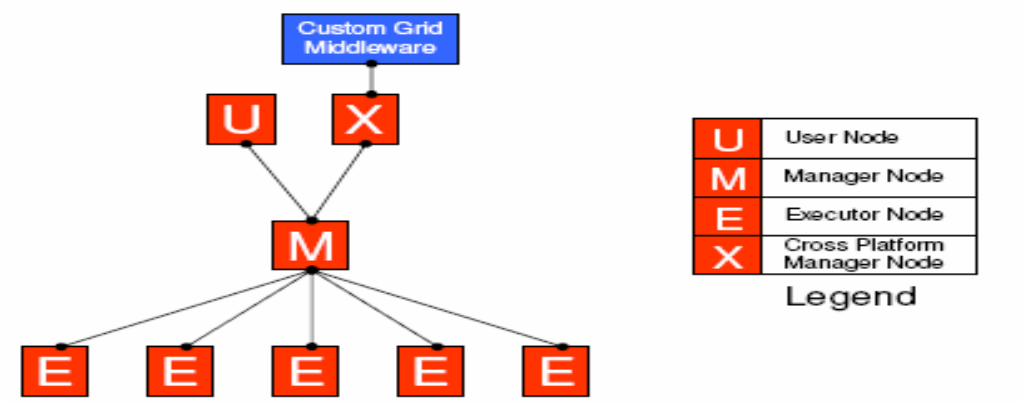


Figure A.1: Distributed Components of Alchemi

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.

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.

A.2.2 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, Alchemi's execution model provides the dual benefit of:

- flexible resource management i.e. centralized management with dedicated execution vs. decentralized management with non-dedicated execution; and
- flexible deployment under network constraints i.e. the component can be deployment as non- dedicated 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 AlchemiGridThread 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.

A.2.3 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 user's 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.

A.2.4 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 interoperate with and leverage Alchemi on any platform that supports web services.

A.3 TECHNICAL DETAILS

The objects remoted using .NET Remoting within the four distributed components of Alchemi, the Manager, Executor, Owner and Cross-Platform Manager are instances of GManager, GExecutor, GApplication and CrossPlatformManager respectively. It should be noted that classes are named with respect to their roles vis-à-vis a grid application. This discussion therefore refers to an 'Owner' node synomymously with a 'User' node, since the node where the grid application is being submitted from can be considered to "own" the application. "The prefix 'I' is used in type names to denote an interface, whereas 'G' is used to denote a 'grid node' class. GManager, GExecutor, GApplication derive from the GNode class which implements generic functionality for remoting the object itself and connecting to a remote Manager via the IManager interface.

The Manager executable initializes an instance of the GManager class, which is always remoted and exposes a public interface IManager. The Executor executable creates an instance of the GExecutor class. For nondedicated execution, there is one-way communication between the Executor and the Manager. Where twoway communication is possible and dedicated execution is desired, GExecutor is remoted and exposes the IExecutor interface so that the Manager may communicate with it directly. The Executor installation provides an option to install a screen saver, which initiates non-dedicated execution when activated by the operating system.

The GApplication object in Alchemi API communicates with the Manager in a similar fashion to GExecutor. While two-way communication is currently not used in the implementation, the architecture caters for this by way of the IOwner interface.

The Cross-Platform Manager web service is a thin wrapper around GManager and uses applications and threads internally to represent tasks and jobs (the GJob class derives from GThread) via the public ICrossPlatformManager interface.

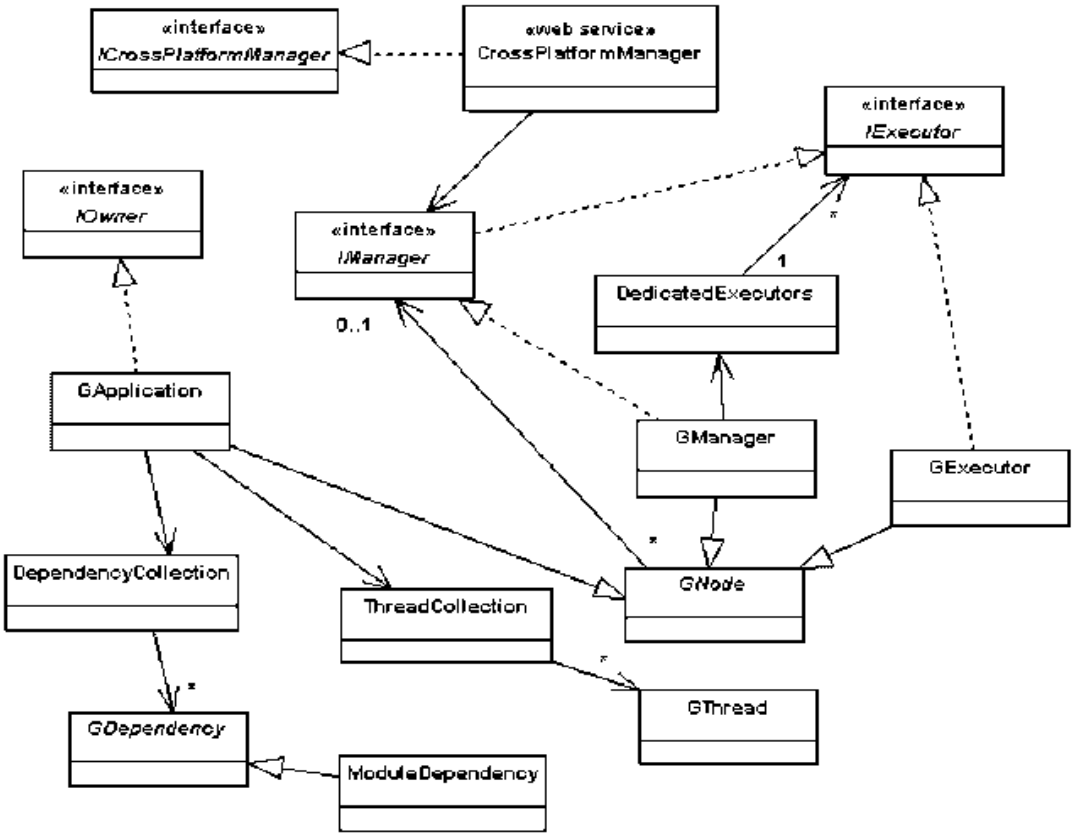


Figure A.2: Main Classes and Relationship in Alchemi

OGSI.NET B

B.1 INTRODUCTION

The core of OGSA has been the Open Grid Services Infrastructure (OGSI), which essentially defines the common capabilities and interfaces of the "hosting environment" for services that adhere to the larger vision of the OGSA. OGSI defines conventions for creating, managing, and exchanging information between Grid services. The Open Grid Service Infrastructure (OGSI) has been designed to facilitate the creation of multiple, interoperable Grid Service hosting environments OGSI.NET is built upon the .NET Framework, which is the Web Services infrastructure of Microsoft.

B.2 BASIC ARCHITECTURE

In many ways, the single design goal of OGSI.NET is to achieve OGSI-compliance by exploiting the core technologies of Microsoft as much as possible. Simply, by creating the thinnest layer possible, we could replace current Microsoft technologies with newer versions as they become available.

The basic design of OGSI.NET is to have a container entity that "holds" all service instances running on a host. The container process consists of a collection of ApplicationDomains (or AppDomains), Microsoft's mechanism for intra-process memory protection. Each service instance executes in its own AppDomain and there is one additional domain for the container's logic (some dispatching and message processing functionality). The object in this final AppDomain is referred to as the dispatcher. While the current version of OGSI.NET (2.0) keeps each service in its own AppDomain, there is an open issue about other possible mappings for future versions of the container.

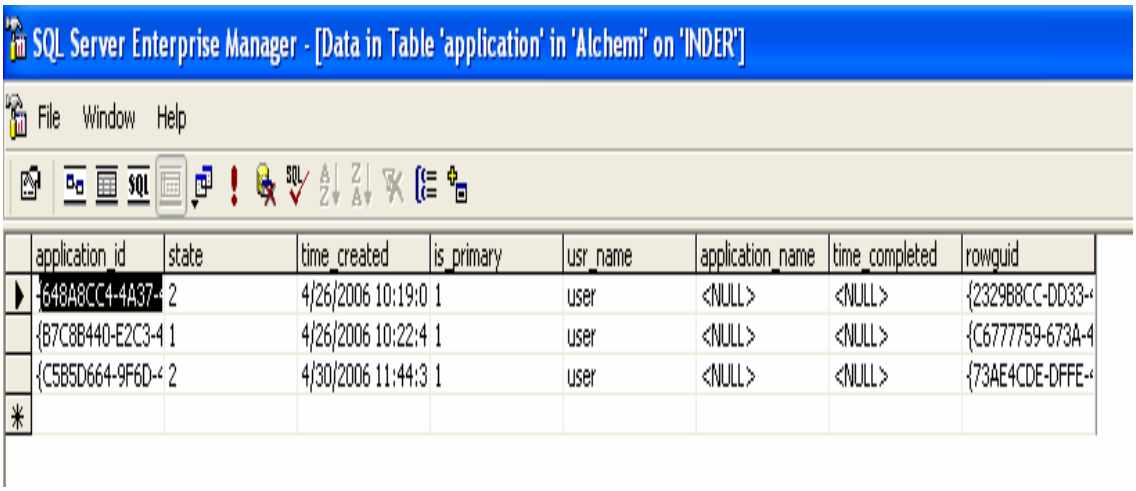
For example, all instances created by a given factory could live in the same AppDomain. Having more than one service in a given AppDomain would make interservice communication more efficient, but at a cost of some security since any errant or malicious service has direct access to the memory of all the services in its AppDomain.

A client makes a request on the OGSI.NET architecture by sending a message to the IIS web server. In order to support arbitrary names for grid services (as required by the OGSI pecification), OGSI.NET uses an ISAPI filter to intercept requests at an early stage in the IIS request chain. This filter rewrites the request so that IIS will dispatch it to OGSI.NET's ASP.NET HttpHandler. This HttpHandler dispatches the request to the OGSI.NET container. The container process has a thread pool and each IIS request causes one of the container process' threads to execute the dispatcher. The dispatcher determines which service instance should get the request and transfers execution of that thread to an object in the appropriate AppDomain.

Inside the AppDomain, control transfers to an object called the Grid Service Wrapper (GSW). A GSW encapsulates a service instance, including the implementations of its methods and its service data. The GSW maintains lists of the port types that the service supports (both custom port types written by the service's author and OGSI specification port types provided with the container, e.g. grid service port type, notification source port type, etc.) and the serializers and deserializers needed for the various messaging protocols the service supports (e.g. SOAP or Microsoft Remoting). The GSW performs processing of any message specific information (e.g. SOAP headers) and deserializes the request message to get the name of the method to invoke and any parameters. Then the GSW selects the port type that implements the requested method and uses reflection to get a handle to the method. The method is invoked with any parameters and message specific data (e.g. SOAP header data) sent in the request. The data returned by the invocation is serialized by the GSW and sent back to the dispatcher as a binary array. The dispatcher sends this array back to IIS for return to the client.

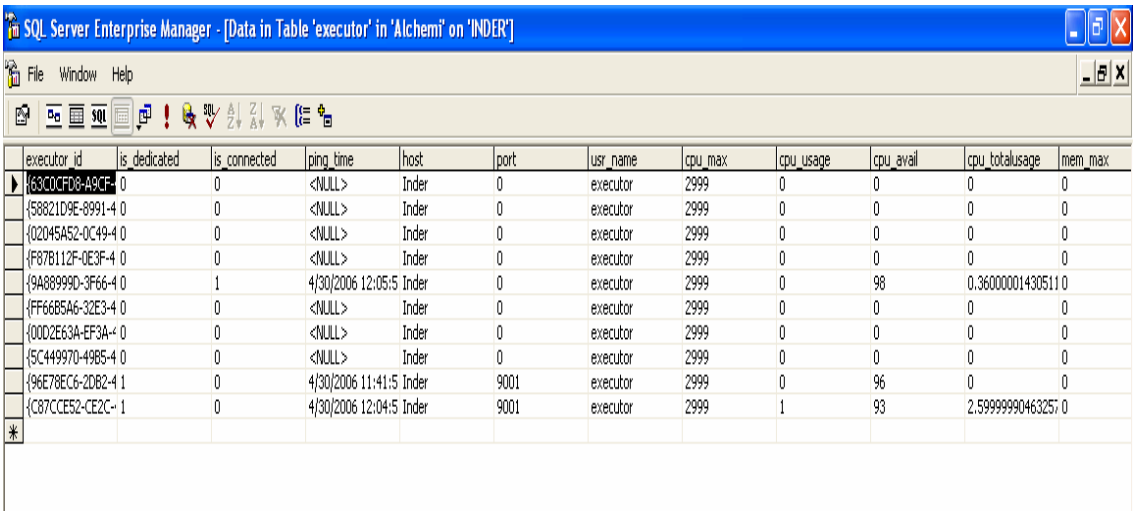
DATABASE STRUCTURE C

The database structure which we replicated to build the backup manager consists of following tables. These tables maintain all the information regarding grid.



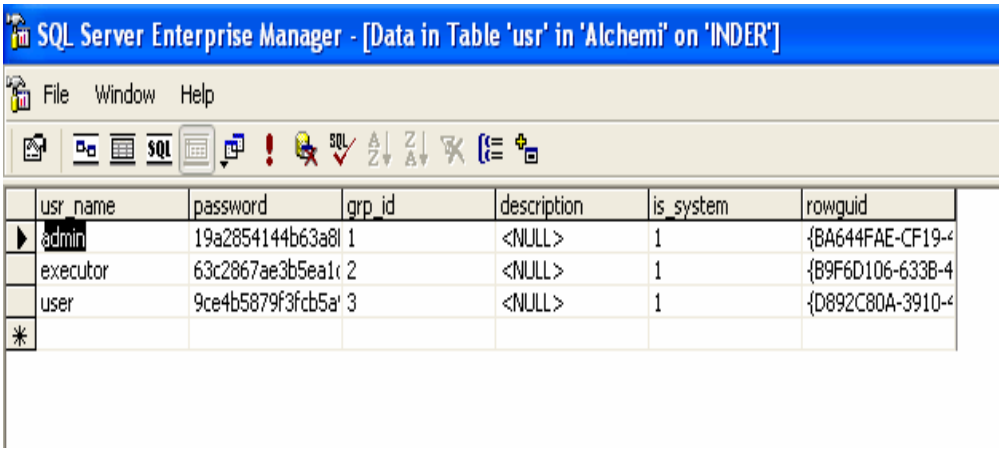
application_id	state	time_created	is_primary	usr_name	application_name	time_completed	rowguid
{648A8CC4-4A37-4	2	4/26/2006 10:19:0	1	user	<NULL>	<NULL>	{2329B8CC-DD33-4
{B7C8B440-E2C3-4	1	4/26/2006 10:22:4	1	user	<NULL>	<NULL>	{C6777759-673A-4
{C5B5D664-9F6D-4	2	4/30/2006 11:44:3	1	user	<NULL>	<NULL>	{73AE4CDE-DFFE-4

Table C.1: Application



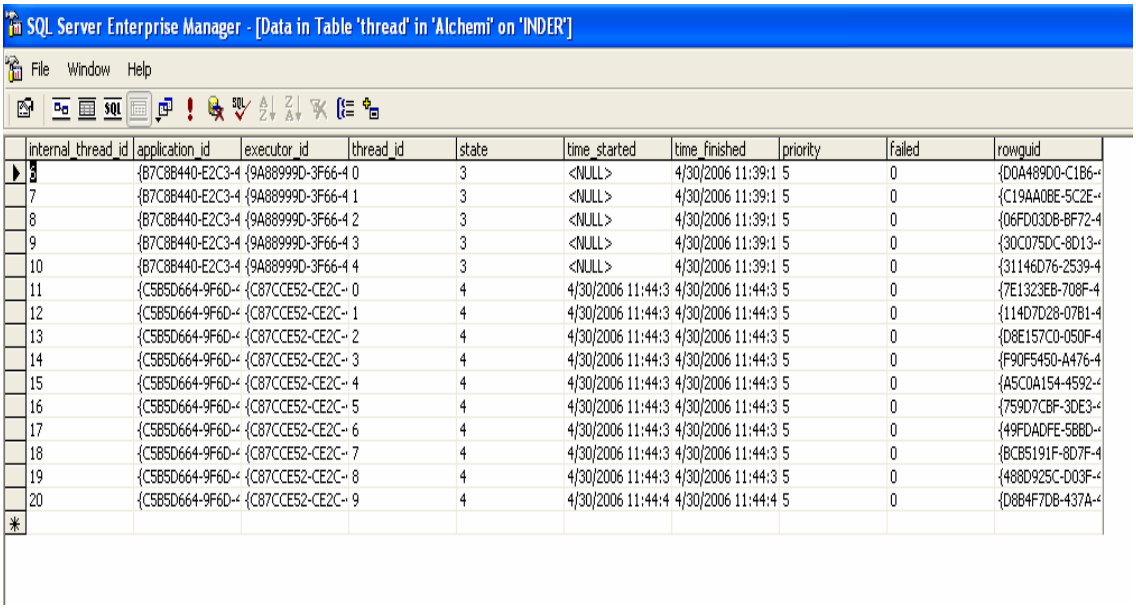
executor_id	is_dedicated	is_connected	ping_time	host	port	usr_name	cpu_max	cpu_usage	cpu_avail	cpu_totalusage	mem_max
{63C0CFD8-A9CF-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{58821D9E-8991-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{02045A52-0C49-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{F87B112F-0E3F-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{9A88999D-3F66-4	0	1	4/30/2006 12:05:5	Inder	0	executor	2999	0	98	0.36000001430511	0
{FF66B5A6-32E3-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{00D2E63A-EF3A-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{5C449970-49B5-4	0	0	<NULL>	Inder	0	executor	2999	0	0	0	0
{96E78EC6-2DB2-4	1	0	4/30/2006 11:41:5	Inder	9001	executor	2999	0	96	0	0
{C87CCE52-CE2C-4	1	0	4/30/2006 12:04:5	Inder	9001	executor	2999	1	93	2.5999999046325	0

Table C.2: Executor



usr_name	password	grp_id	description	is_system	rowguid
admin	19a2854144b63a8l	1	<NULL>	1	{BA644FAE-CF19-4
executor	63c2867ae3b5ea1(	2	<NULL>	1	{B9F6D106-633B-4
user	9ce4b5879f3fcb5a'	3	<NULL>	1	{D892C80A-3910-4

Table C.3: User



internal_thread_id	application_id	executor_id	thread_id	state	time_started	time_finished	priority	failed	rowguid
3	{B7C8B440-E2C3-4	{9A88999D-3F66-4	0	3	<NULL>	4/30/2006 11:39:1	5	0	{D0A489D0-C1B6-4
7	{B7C8B440-E2C3-4	{9A88999D-3F66-4	1	3	<NULL>	4/30/2006 11:39:1	5	0	{C19AA0BE-5C2E-4
8	{B7C8B440-E2C3-4	{9A88999D-3F66-4	2	3	<NULL>	4/30/2006 11:39:1	5	0	{06FD03DB-BF72-4
9	{B7C8B440-E2C3-4	{9A88999D-3F66-4	3	3	<NULL>	4/30/2006 11:39:1	5	0	{30C075DC-8D13-4
10	{B7C8B440-E2C3-4	{9A88999D-3F66-4	4	3	<NULL>	4/30/2006 11:39:1	5	0	{31146D76-2539-4
11	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	0	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{7E1323EB-708F-4
12	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	1	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{114D7D28-07B1-4
13	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	2	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{D8E157C0-050F-4
14	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	3	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{F90F5450-A476-4
15	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	4	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{A5C0A154-4592-4
16	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	5	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{759D7CBF-3DE3-4
17	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	6	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{49FDADFE-5BBD-4
18	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	7	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{BCB5191F-8D7F-4
19	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	8	4	4/30/2006 11:44:3	4/30/2006 11:44:3	5	0	{488D925C-D03F-4
20	{C5B5D664-9F6D-4	{C87CCE52-CE2C-4	9	4	4/30/2006 11:44:4	4/30/2006 11:44:4	5	0	{D8B4F7DB-437A-4

Table C.4: Thread

PAPERS ACCEPTED

1. Inderpreet Chopra and Inderveer Chana " A Taxonomy and Survey of Resource Management in Grid Middleware" at Las Vegas, **GCA'06** - The 2006 International Conference on Grid Computing and Applications, Monte Carlo Resort, Las Vegas, Nevada, USA (June 26-29, 2006)

Subject: GCA'06 - Your Paper
From: "GCA'06 Conference" <gca@pixel.cviog.uga.edu>
Date: Tue, April 4, 2006 7:30 pm
To: inderpreet.chopra@tiet.ac.in (more)
Priority: Normal
Options:

Dear Drs. Inderpreet Chopra and Inderveer Chana:

I am pleased to inform you that the following paper which you submitted to The 2006 International Conference on Grid Computing and Applications (GCA'06: June 26-29, 2006, Las Vegas, USA) has been accepted as a Regular Research Report (RRR); Please refer to the appended information for the categories of accepted papers.

Paper ID #: GCA4824
Title: A Taxonomy and Survey of Resource Management in Grid Middleware
Inderpreet Chopra and Inderveer Chana

Each paper was refereed by at least two reviewers. Your paper was evaluated by the editorial board chaired by Dr. R. J. Kumar. Comments (if any) from the reviewers will be emailed to you on April 6 (you would receive comments if and only if the referees determined that some parts of the paper will have to be changed.) If you do not receive any comments by April 6th, then you must assume that the referees had requested no mandatory changes to the paper.

Appended to this email message, you will find the following information / instructions:

1. Important Dates (deadlines)
2. Conference Registration
3. Typing Instructions for the Preparation of the Final Camera-Ready Papers
4. Submission of Final Camera-Ready Papers for publication in the conference proceedings (hardcopy book) and inclusion in CD-ROM.
5. Hotel Reservation
6. Invitation Letters for US Visa Purposes
7. Conference Program/Schedule
8. Presentation Formats / Accepted Paper Categories

Please note that the conference will not be mailing you any conference materials. Your conference registration package will be available for pick up at the conference site in Las Vegas. You are encouraged to pickup your registration package on Sunday June 25, 2006 (after 3:00 PM) at the conference registration desk.

Congratulations, and thank you for your contribution to the Conference. I look forward to seeing you at the conference in Las Vegas (June 26-29, 2006).

Kind regards,
Hamid R. Arabnia, PhD
Coordinator, WORLDCOMP'06
hra@cs.uga.edu

2. Inderpreet Chopra and Inderveer Chana " Resource Management in Grid Middleware" at JMIT Radaur, ICT 06 (February 9-11, 2006)

Subject: Regarding National Seminar ICT-2006 JMIT Radaur
From: "Gaurav Sharma" <gaurav13@gmail.com>
Date: Sun, February 5, 2006 6:39 pm
To: raakeshdhiman@gmail.com
Priority: Normal
Options:

Hello Sir/Madam,

Your paper has selected and I am sending you Tentative Schedule for AICTE sponsored National Seminar on "Information & Communication Technology Recent Advances & Applications" as attachment(ZIP). At the time of registration you have to submit your Talk/Paper, so please bring a hard-copy & a soft-copy (C.D.) of your Talk/Paper, in the already mentioned format, so that we can include your Talk/Paper in the proceedings without any error. Kindly confirm your arrival Date & Time at radaur.

Thanks with

Regards

Gaurav Sharma

9896091772

3. Inderpreet Chopra, Inderveer Chana, "Fault Tolerance in Grid Middleware" at IIT Guwahati, 8th International Conference on Distributed Computing and Networking, ICDCN 2006, (December 27-30, 2006) [Communicated]