Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"A Proactive Fault Tolerance Approach for Grid Environment"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of 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.

(Puneet Khurana)

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)
Supervisor
CSED
Thapar University
Patiala

Countersigned by:

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

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

Acknowledgement

I wish to express my deep gratitude to my guide Ms. Inderveer Chana, Senior Lecturer, Computer Science & Engineering Department for providing her uncanny guidance and support throughout the preparation of the thesis report.

I am also heartily thankful to and Dr. (Mrs.) Seema Bawa Professor, Computer Science and Engineering Department & Head, Mr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department and Mrs. Shivani Goel, P.G. Coordinator, Computer Science and Engineering Department for the motivation and inspiration that triggered me for my thesis work.

I would also like to thank all the staff members, T.I.E.T. Grid Group and all my friends especially Anurag, Ratnesh, Santosh, Ms. Anju Sharma and Ms. Shashi who were always there at the need of the hour and provided all the help and support, which I required for the completion of the thesis.

Last but not the least, I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Puneet Khurana
(8053117)

Abstract

Grid technology has emerged as a new way of large-scale distributed computing with high-performance orientation. Grid computing tries to bring, under one definitional umbrella all the work being done in the high performance, cluster, peer-to-peer, and Internet computing arenas. Grid computing solutions are constructed using a variety of technologies and open standards. Grid computing, in turn, provides highly scalable, highly secure, and extremely high-performance mechanisms for discovering and negotiating access to remote computing resources in a seamless manner. This makes it possible for the sharing of computing resources, on an unprecedented scale, among an infinite number of geographically distributed groups. The increasing complexity of Grid services and systems demands correspondingly larger human effort for system configuration and performance management. The management issues, which are mainly done in a manual style today, become time-consuming, error-prone and even unmanageable for human administrators.

The fault tolerance is the major area of concern in the self-management of the Grid.

The terms self-management or self-healing have been borrowed from Autonomic Computing. Autonomic Computing helps to address complexity by using technology to manage technology

In this work, a system for the coordinated, Autonomic management (self-healing) of multiple clusters in a Grid has been discussed. The clusters are integrated into the Grid making the Grid Fault tolerant. The existing fault tolerance mechanism in the Grid uses the reactive approaches. The focus of the thesis is to design a prototype for proactive fault tolerance in Grid environment.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

In the recent years, Grid has facilitated the sharing and integration of large scale, heterogeneous resources and has been widely recognized as the future framework of Distributed Computing.

This chapter describes the challenges in Grid Computing and gives the overview of the work done in the thesis.

1.1 Grid Computing

Grid is standards based application/resource sharing architecture that makes it possible for heterogeneous systems and applications to share, compute and storage resources transparently [1]. The Grid infrastructure promises seamless access to computational and storage resources, and offers the possibility of cheap, ubiquitous Distributed Computing [2]. Grid technology is having a fundamental impact on the economy by creating new areas, such as e-Government and e-Health, new business opportunities, such as computational and data storage services, and changing business models, such as greater organizational and service devolution. The Grid historically arose out of a need to perform massive computation; the current direction demonstrates the potential to change the structure of electronic service provision and create a new Grid service economy. The success of the Grid will be founded on the development of new Grid-enabled software systems and the evolution of legacy systems to Grid-enabled systems. Over time Grid Computing will enable a more flexible, efficient and utility-like global Computing infrastructure. The key to realizing the benefits of Grid Computing is standardization and self-management, so that the diverse resources that make up a modern Computing environment can be discovered, accessed, allocated, monitored, and in general managed as a single virtual system—even when provided by different vendors and/or operated by different organizations.

1.2 Challenges with Grid

Grid is getting complex day by day. The increasing complexity of Grid services and systems demands correspondingly larger human effort for system configuration and performance management. The management issues, which are mainly done in a

manual style today, become time consuming, error-prone and even unmanageable for human administrators. The major barriers in the management of Grid are [3]:

- *Complexity*:

The Grid is complex in nature because it tries to couple large-scale disparate, distributed and heterogeneous resources – such as data, computers, operating systems, database systems, applications and special devices – which may run across multiple virtual organizations to provide a uniform Computing platform.

- *Dynamic nature*:

The Grid is a dynamic Computing environment in that resource and services can join and leave at any time. The time and cost required in managing a Grid is much more than setting up the Grid itself. A fundamental challenge is creating a Grid which is capable of managing itself. In other words, challenge is in creating correct, robust, flexible and cost-effective Grid-enabled software. The main challenge in making Grid self managing is in making it fault-tolerant.

Because of computational Grid heterogeneity, scale and complexity, faults become likely. Therefore, Grid infrastructure must have mechanisms to deal with faults while also providing efficient and reliable services to its end users.

To make a Grid Fault Tolerant, Grid should able to handle the faults proactively and should heal itself if the faults have occurred. In other words Grid should be self-healing. The term Self-Healing is one of the basic characteristic of Autonomic Computing.

Autonomic Computing concept was first presented and advocated by IBM [4]. The vision of Autonomic Computing is to design and build Computing systems that possess inherent self-managing capabilities. Each Autonomic system is a collection of Autonomic elements, which are elementary constituents that possess services and resources. By self-configuration, self-optimization, self-healing, and self-protection, Autonomic Computing can improve the level of automation and self-management capability to a far greater extent than it is today in Grid Computing systems.

The goal of Self-Managing systems is to dramatically reduce the cost and perceived complexity of the Grid by enabling it to manage itself in accordance with high-level guidance from humans. In the thesis work, the terms self-healing and self-managing has been used interchangeably as self-healing is one of the main characteristic of self-management.

1.3 Motivation

The management of Grid is a daunting task and the time-cost factor is the major concern. The self-organized model of Autonomic Computing is more suitable for Grid environments, which is characterized by distributed, open and dynamic properties. The motivation is to provide the Grid, the Fault tolerant capabilities so that the management tasks can be carried out with minimum human intervention keeping time-cost intact.

1.4 Organization of Thesis

The chapters in the thesis are organized as follows:

Chapter 2 Describes in detail what Grid Computing is, Grid characteristics, and various types of Grid systems, Grid architecture, OGSA, and its related organizations

Chapter 3 Describes the Fault tolerance in Grid, Autonomic System, its characteristics and its architecture. The monitoring system as specified in OGSA specification and various other monitoring mechanisms have been discussed.

Chapter 4 Discusses the problems in the existing Grid fault tolerance and health check mechanism approaches.

Chapter 5 Includes the architectural details of the proactive fault tolerant cluster in Grid Fabric and its design details.

Chapter 6 Discusses the Implementation details of the system prototype.

Chapter 7 Summarizes the work presented in this thesis followed by the features that can be incorporated in future for the enhancement of the fault tolerance in the Grid environment.

Chapter 2 The Grid Computing

This chapter presents the detailed description of the Grid and Grid Computing concepts, characteristics of Grids, Grid Architecture and the organizations associated to the Grid and their roles.

2.1 Introduction

Grid Computing environments have proven to be incredibly significant in the today's pervasive world of needing information anytime and anywhere such that they are often referred to as being the world's single and most powerful computer solutions. It has been realized that with the many benefits of Grid Computing, both a complicated and complex global environment have been consequently introduced, which leverages a multitude of open standards and technologies in a wide variety of implementation schemes.

Grid Computing solutions are constructed using a variety of technologies and open standards [5]. Grid Computing, in turn, provides highly scalable, highly secure, and extremely high-performance mechanisms for discovering and negotiating access to remote computing resources in a seamless manner. This makes it possible for the sharing of computing resources, on an unprecedented scale, among an infinite number of geographically distributed groups. This serves as a significant transformation agent for individual and corporate implementations surrounding computing practices, toward a general-purpose utility approach very similar in concept to providing electricity or water.

2.2 Grid Computing Definitions

Grid Computing tries to bring, under one definitional umbrella all the work being done in the high performance, cluster, peer-to-peer, and Internet computing arenas. Coming up with a definition for Grid Computing, therefore is not as easy as one would have expected. Some of the definitions of Grid Computing that have been uncovered include [6]:

- The flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources.
- Transparent, secure, and coordinated resource sharing and collaboration across sites.

- The ability to form virtual, collaborative organizations that share applications and data in an open heterogeneous server environment in order to work on common problems.
- The ability to aggregate large amounts of computing resources which are geographically dispersed to tackle large problems and workloads as if all the servers and resources are located in a single site.
- A hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to computational resources.
- The Web provides us information—the Grid allows us to process it.
- A Grid is a software framework providing layers of services to access and manage distributed hardware and software resources.

Although some of these definitions are quite good, the following broader definition of Grid Computing serves the purpose of this book more fully and will be used to describe Grid systems:

"*Grid Computing enables virtual organizations to share geographically distributed resources as they pursue common goals, assuming the absence of central location, central control, omniscience, and an existing trust relationship"* [7].

2.3 The Concept of Virtual Organizations

The concept of a virtual organization is the key to Grid Computing [8]. It is defined as a dynamic set of individuals and/or institutions defined around a set of resource-sharing rules and conditions. All these virtual organizations share some commonality among them, including common concerns and requirements, but may vary in size, scope, duration, sociology, and structure.

The members of any virtual organization negotiate on resource sharing based on the rules and conditions defined in order to share the resources from the thereby automatically constructed resource pool. Assigning users, resources, and organizations from different domains across multiple, worldwide geographic territories to a virtual organization are one of the fundamental technical challenges in Grid Computing. This complexity includes the definitions of the resource discovery mechanism, resource sharing methods, rules and conditions by which this can be achieved, security federation and/or delegation, and access controls among the participants of the virtual organization. This challenge is both complex and complicated across several dimensions.

Virtual organizations can span from small corporate departments that are in the same physical location to large groups of people from different organizations that are spread out across the globe. Virtual organizations can be large or small, static or dynamic [9].

2.4 Characteristics of Grids

There are many issues in computational Grid as depicted in figure 2.1 but there are three main issues that characterize computational Grids [10]:

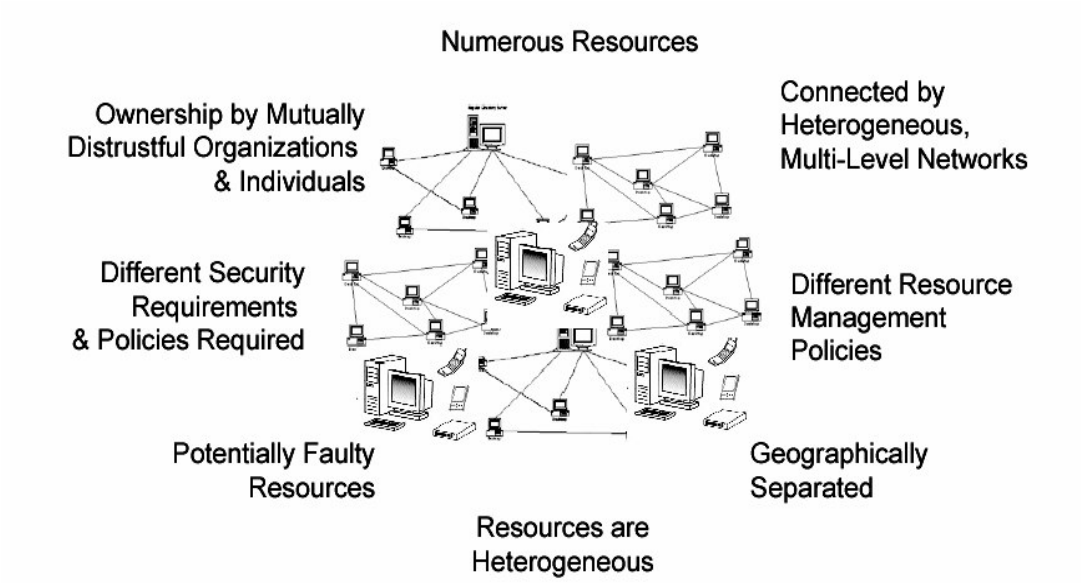


Figure 2.1 Characteristics of Grids [10].

- **Heterogeneity**: A Grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.
  - Resources are heterogeneous
  - Resources are administratively disparate
  - Resources are geographically disparate
  - Users do not have to worry about system details (e.g., location, operating system, accounts).
  - Resources are numerous.
  - Resources have different resource management policies.
  - Resources are owned and managed by different, potentially mutually distrustful organizations and individuals that likely have different security policies and practices.

- **Scalability**: Generic definition of scalability can be given as ability of a solution to some problem (computer application or product) to continue to function well as the problem (or its context) increases in size or volume. In term of Grid it 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 behavior dynamically so as to extract the maximum performance from the available resources and services.

2.5 Grid Computing Model

New architecture model and technology has been developed for the establishment, management, and cross-organizational resource sharing within a virtual organization. This new architecture, called Grid architecture, identifies the basic components of a Grid system, defines the purpose and functions of such components and indicates how each of these components interacts with one another. The main attention of the architecture is on the interoperability among resource providers and users to establish the sharing relationships. This interoperability means common protocols at each layer of the architecture model, which leads to the definition of a *Grid protocol architecture* [11]. This protocol architecture defines common mechanisms, interfaces, schema, and protocols at each layer, by which users and resources can negotiate, establish, manage, and share resources.

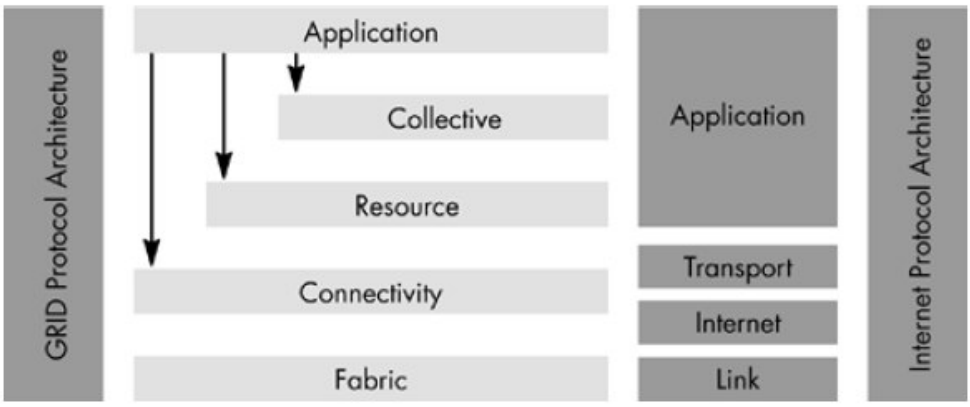


Figure 2.2 The Grid Architecture [11].

2.5.1 Fabric Layer: Interface to Local Resources

The Fabric layer defines the resources that can be shared. These resources can be physical resources or logical resources by nature.

Typical examples of the logical resources found in a Grid Computing environment are distributed file systems, computer clusters, distributed computer pools, software applications, and advanced forms of networking services. These logical resources are implemented by their own internal protocol (e.g., network file systems [NFS]) for distributed file systems, and clusters using logical file systems [LFS]). These resources then comprise their own network of physical resources.

2.5.2 Connectivity Layer: Manages Communications

The Connectivity layer defines the core communication and authentication protocols required for Grid-specific networking services transactions. Communications protocols, which include aspects of networking transport, routing, and naming, assist in the exchange of data between fabric layers of respective resources. The authentication protocol builds on top of the networking communication services in order to provide secure authentication and data exchange between users and respective resources.

For communication services, the most commonly used Network layer protocol is the TCP/IP Internet protocol stack. The authentication solution for virtual organization environments requires significantly more complex characteristics. The following describes the characteristics for consideration:

- *Single sign-on*: This provides any multiple entities in the Grid fabric to be authenticated once; the user can then access any available resources in the Grid Fabric layer without further user authentication intervention.
- *Delegation:* This provides the ability to access a resource under the current user's permissions set; the resource should be able to relay the same user credentials (or a subset of the credentials) to other resources respective to the chain of access.
- *Integration with local resource specific security solutions*: Each resource and hosting has specific security requirements and security solutions that match the local environment.
- *User-based trust relationships*: In Grid Computing, establishing an absolute trust relationship between users and multiple service providers is very critical.

- *Data security*: The data passing through the Grid Computing solution, no matter what complications may exist, should be made secure using various cryptographic and data encryption mechanisms.

2.5.3 Resource Layer: Sharing of a Single Resource

The Resource layer utilizes the communication and security protocols defined by the networking communications layer, to control the secure negotiation, initiation, monitoring, metering, accounting, and payment involving the sharing of operations across individual resources. This layer only handles the individual resources and, hence, ignores the global state and atomic actions across the other resource collection, which in the operational context is the responsibility of the Collective layer.

There are two primary classes of resource layer protocols. These protocols are the key to the operations and integrity of any single resource. These protocols are: as follows:

- *Information Protocols*: These protocols are used to get information about the structure and the operational state of a single resource, including configuration, usage policies, service-level agreements, and the state of the resource.
- *Management Protocols*: The important functionalities provided by the management protocols are:
  - Negotiating access to a shared resource is paramount. These negotiations can include the requirements on quality of service, advanced reservation, scheduling, and other key operational factors.
  - Performing operation(s) on the resource, such as process creation or data access, is also a very important operational factor.
  - Acting as the service/resource policy enforcement point for policy validation between a user and resource is critical to the integrity of the operations.
  - Providing accounting and payment management functions on resource sharing is mandatory.
  - Monitoring the status of an operation, controlling the operation including terminating the operation, and providing asynchronous notifications on operation status, is extremely critical to the operational state of integrity.

2.5.4 The Collective Layer: Coordinating Multiple Resources

While the Resource layer manages an individual resource, the Collective layer is responsible for all global resource management and interaction with a collection of resources. This layer of protocol implements a wide variety of sharing behaviors

(protocols) utilizing a small number of Resource layer and Connectivity layer protocols. Some key examples of the common, more visible collective services in a Grid Computing system are as follows:

- *Discovery Services*: This enables the virtual organization participants to discover the existence and/or properties of that specific available virtual organization's resources.
- *Co-allocation, Scheduling, and Brokering Services*: These services allow virtual organization participants to request the allocation of one or more resources for a specific task, during a specific period of time, and to schedule those tasks on the appropriate resources.
- *Monitoring and Diagnostic Services*: These services afford the virtual organizations resource failure recovery capabilities, monitoring of the networking and device services, and diagnostic services that include common event logging and intrusion detection.
- *Data Replication Services*: These services support the management aspects of the virtual organization's storage resources in order to maximize data access performance with respect to response time, reliability, and costs.
- *Grid-Enabled Programming Systems*: These systems allow familiar programming models to be utilized in the Grid Computing environments, while sustaining various Grid Computing networking services.
- *Workload Management Systems and Collaborative Frameworks*: This provides multi-step, asynchronous, multi-component workflow management.
- *Software Discovery Services*: This provides the mechanisms to discover and select the best software implementation(s) available in the Grid environment, and those available to the platform based on the problem being solved.
- *Community Authorization Servers*: These servers control resource access by enforcing community utilization policies and providing these respective access capabilities by acting as policy enforcement agents.
- *Community Accounting and Payment Services*: These services provide resource utilization metrics, while at the same time generating payment requirements for members of any community.

2.5.5 Application Layer: User-Defined Grid Applications

These are user applications, which are constructed by utilizing the services defined at each lower layer. Such an application can directly access the resource, or can access

the resource through the Collective Service interface APIs (Application Provider Interface).

Each layer in the Grid architecture provides a set of APIs and SDKs (software developer kits) for the higher layers of integration. It is up to the application developers whether they should use the collective services for general-purpose discovery, and other high-level services across a set of resources, or if they choose to start directly working with the exposed resources. These user-defined Grid applications are (in most cases) domain specific and provide specific solutions.

2.6 Open Grid Service Architecture (OGSA)

The job of the OGSA is to build on the Grid service specification (Open Grid Service Infrastructure, or OGSI) to define architectures and standards for a set of "core Grid services" that are essential components to every Grid [47]. OGSA architecture is a layered architecture, with clear separation of the functionalities at each layer. The core architecture layers are OGSI, which provides the base infrastructure, and OGSA core platform services, which are a set of standard services including policy, logging, service-level management, and so on. The high-level applications and services use these lower layer core platform components and OGSI that become part of a resource-sharing Grid.

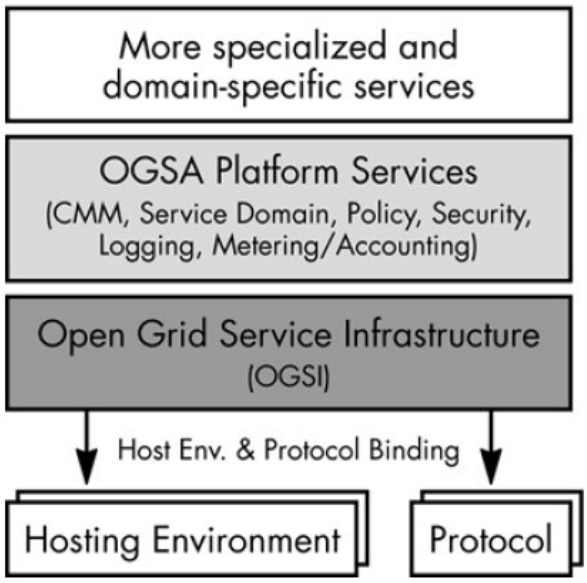


Figure 2.3 OGSA Platform Architecture [12].

The major OGSA goals are [12]:

- Identify the use cases that can drive the OGSA platform components
- Identify and define the core OGSA platform components

- Define hosting and platform-specific bindings
- Define resource models and resource profiles with interoperable solutions

In addition to the broad goals defined above, OGSA defines more specific goals, including [13]:

- Facilitating distributed resource management across heterogeneous platforms
- Providing seamless quality of service delivery
- Building a common base for Autonomic management solutions
- Providing common infrastructure building blocks to avoid "stovepipe solution towers"
- Open and published interfaces and messages
- Industry-standard integration solutions including Web services
- Facilities to accomplish seamless integration with existing IT resources where resources become on-demand services/resources
- Providing more knowledge-centric and semantic orientation of services

2.7 Types of Grids

Grid Computing vendors have adopted various nomenclatures to explain and define the different types of Grids. Some define Grids based on the structure of the *organization* (virtual or otherwise) that is served by the Grid, while others define it by the principle *resources* used in the Grid. In this section we attempt to classify these varying definitions [14].

2.7.1 Departmental Grids

Departmental Grids are deployed to solve problems for a particular group of people within an enterprise. The resources are not shared by other groups within the enterprise [15, 48]. Following is a list of vendor definitions that are believed to refer to departmental Grids.

- *Cluster Grids*: Cluster Grid is a term used by Sun Microsystems and consists of one or more systems working together to provide a single point of access to users. It is typically used by a team for a single project and can be used to support both high throughput and high performance jobs.
- *Infra Grids*: Infra Grid is a term used by IBM to define a Grid that optimizes resources within an enterprise and does not involve any other internal partner. It can be within a campus or across campuses.

2.7.2 Enterprise Grids

Enterprise Grids consist of resources spread across an enterprise and provide service to all users within that enterprise [16, 48]. The following vendor definitions fall into this category.

- *Enterprise Grids*: An enterprise Grid, according to Platform Computing, is deployed within large corporations that have a global presence or a need to access resources outside a single corporate location. Enterprise Grids run behind the corporate firewall.
- *Intra Grids*: According to IBM, resource sharing among different groups within an enterprise constitutes an intra Grid. An intra Grid can be local or traverse the wide area network. Intra Grids are located within the corporate firewall.
- *Campus Grids*: Campus Grids, according to Sun Microsystems, enable multiple projects or departments to share computing resources in a cooperative way. Campus Grids may consist of dispersed workstations and servers as well as centralized resources located in multiple administrative domains, in departments, or across the enterprise.

2.7.3 Extraprise Grids

Extraprise Grids are established between companies, their partners, and their customers. The Grid resources are generally made available through a virtual private network [17, 48]. Following are some of the terms used by various vendors to describe such Grids.

- *Extra Grids*: Extra Grids, according to IBM, enable sharing of resources with external partners. This assumes that connectivity between the two enterprises is through some trusted service, such as a private network or a virtual private network.
- *Partner Grids*: Platform Computing defines these as Grids between organizations within similar industries, which have a need to collaborate on projects and use each other's resources as a means to reach a common goal.

2.7.4 Global Grids

Grids established over the public Internet constitute global Grids. They can be established by organizations to facilitate their business or purchased in part, or in whole, from service providers [18, 48]. Following are some vendor definitions that fall in this category.

- *Global Grids*: Global Grids, as defined by Sun, allow users to tap into external resources. Global Grids provide the power of distributed resources to users anywhere in the world for computing and collaboration. They can be used by individuals or organizations to send overflow work over the public network to a Grid services provider.
- *Inter Grids*: Inter Grids, according to IBM, provide the ability to share compute and data/storage resources across the public Web. This can involve sharing resources with other enterprises or buying or selling of excess capacity.

2.7.5 Compute Grids

Compute Grids are created solely for the purpose of providing access to computational resources [19, 48]. Compute Grids can be further classified by the type of computational hardware deployed.

- *Desktop Grids*: These are Grids that leverage the compute resources of desktop computers. Because of the true (but unfortunate) ubiquity of Microsoft® Windows® operating system in corporations, desktop Grids are assumed to apply to the Windows environment. The Mac OS™ environment is supported by a limited number of vendors.
- *Server Grids*: Some corporations, while adopting Grid Computing, keep it limited to server resources that are within the purview of the IT department. Special servers, in some cases, are bought solely for the purpose of creating an internal "utility Grid" with resources made available to various departments. No desktops are included in server Grids. These usually run some flavor of the Unix/Linux operating system.
- *High-Performance/Cluster Grids*: These Grids constitute high-end systems, such as supercomputers or HPC clusters. These are discussed in more detail in later chapters.

2.7.6 Data Grids

Grid deployments that require access to, and processing of, data are called data Grids. They are optimized for data-oriented operations. Although they may consume a lot of storage capacity, these Grids are not to be confused with storage service providers.

2.7.7 Utility Grids

Utility Grids have been defined as being commercial compute resources that are maintained and managed by a service provider. Customers that have the need to

augment their existing, internal computational resources may purchase "cycles" from a utility Grid. In addition to overflow applications, customers may choose to use utility Grids for business continuity and disaster recovery purposes.

2.7.8 Service Grids

Service Grids—as defined by Platform Computing—provide access to resources that can be purchased by corporations to augment their own resources.

2.8 Grid Application Characteristics

The success of Grid Computing will ultimately be determined *not* by the technical sophistication of the Grid Computing protocols, nor by the elegance of Grid networks, but by *what* problems Grid Computing will solve.

Not all applications will be able to take advantage of Grid Computing. The grade of parallel efficiency exhibited by an application determines its suitability for Grid Computing deployment. Parallelism is an inherent property of the application. Parallel applications fall under three general categories [18]:

- *Perfect Parallelism*—Also known as *embarrassingly parallel*. An application can be divided into sets of processes that require little or no communication. A Monte Carlo simulation falls into this category.
- *Data Parallelism*—The same operation is performed on many data elements simultaneously. An example would be using multiple processes to search different parts of a database for one specific query.
- *Functional Parallelism*—Often called *control parallelism*. Multiple operations are performed simultaneously, with each operation addressing a particular part of the problem. For example, in a power plant simulation, one application process might simulate the cool system, one the generator, etc. Data communication from one operation to another is required.

The ratio of the amount of computation done by an application to the amount of communication is called *granularity*. Granularity can be *fine grained*, *medium grained*, or *coarse grained*. In fine-grained tasks, the operands are small and the communication is high. In coarse-grained tasks, entire programs can be executed on a processor before communication or synchronization is required. Obviously, coarse-grained parallel applications generate the best speedup results.

The amount of data being used by each concurrent task, assuming that the application exhibits parallelism, will determine whether a particular application is suited for being

deployed on a desktop Grid, high performance Grid, or cluster Grid. In some cases, desktops may not have the amount of memory required for a particular task or that the network bandwidth requirements may interfere with normal business operations. In each of these cases, applications can then be run on a high-performance Grid that is either in-house or available through a utility computing model.

2.9 Grid Computing Organizations and their Roles

Grid Computing organizations and their roles can be broadly classified into four categories based on their functional role in Grid Computing [19, 50].

Figure 2.4 The Basic Classifications of Grid Computing Organizations [19].

These roles are best described as:

- Organizations developing Grid standards and best practices guidelines
- Organizations developing Grid Computing toolkits, frameworks, and middleware solutions
- Organizations building and using Grid-based solutions to solve their computing, data, and network requirements
- Organizations working to adopt Grid concepts into commercial products, via utility computing, and Business on Demand computing.

2.10 Benefits of Grid Computing

These are the few benefits that Grid Computing provides to user community, developer community and enterprise community as well [19] .It provides an abstraction for resource sharing and collaboration across multiple administrative domains.

Grid Computing enables and manages

- Coordinated sharing and brokering of resources

- Collaborative problem-solving
- Dynamic, virtual communities.

Grid Computing arose out of the need for more cost effective High Performance Computing (HPC) solutions to address critical problems in science and engineering. The initial adoption of the Grid by commercial enterprises has continued to focus on HPC because of the high return on investment and competitive advantage realized by solving compute intensive problems that were previously insolvable in a reasonable period of time or cost.

This section describes the key benefits of a Grid environment:

2.10.1 Exploit 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 7 hours a day and that 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 usages. 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.

2.10.2 Increase Computation

To provide users with more computational power, 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 are fully utilized at that 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.

This chapter described in detail what the Grid Computing is, characteristics of Grid Computing, various kinds of Grids based on the kind of service they provide, applications that can be run on Grid, and benefits of Grid environment.
The next chapter discusses the existing fault tolerance in Grid, concept of Autonomic Computing and the various kind monitoring systems.

Chapter 3 Fault Tolerance and Autonomic Computing

This chapter discusses the Fault Tolerance at cluster level in the Grid, Concept of Autonomic Computing, its architecture and its characteristics. The monitoring system as per OGSA specification and the extended health check mechanism (monitoring) has been discussed.

3.1 Fault Tolerance Mechanisms in Grid

Because of computational Grid heterogeneity, scale and complexity, faults become likely. Therefore, Grid infrastructure must have mechanisms to deal with faults while also providing efficient and reliable services to its end users [20].

3.1.1 Fault Tree Analysis

The fault tree analysis classifies faults that may take place in Grid Computing [21]. In the figure 3.1 various kinds of faults that can occur have been shown.

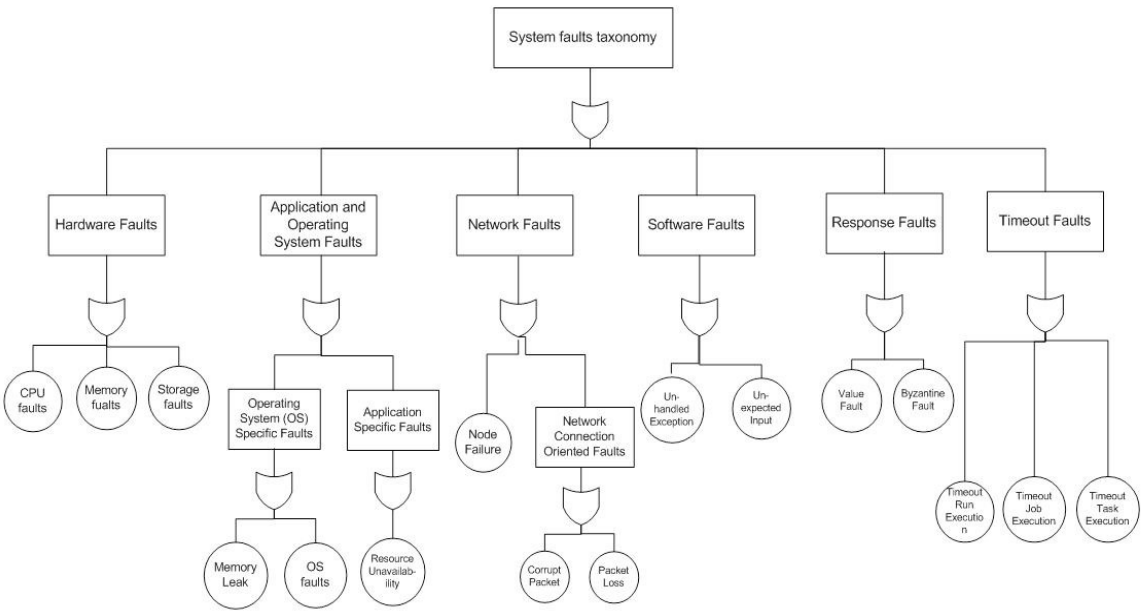


Figure 3.1 Fault Tree Analysis in Grid Computing [21].

There are mainly six classes of faults as discussed below:

1) Hardware faults:

Hardware failures take place due to faulty hardware components such as CPU, memory, and storage devices.

- 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.

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) 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. 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.

4) Software faults:

Several high resource intensive applications usually run 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

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.

6) 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.

3.1.2 Agent Oriented Fault-Tolerant Proactive Approach for Grid

Existing fault-tolerant approaches are inefficient because they are reactive and incomplete [22]. They are reactive because they only deal with faults when they take place; they are incomplete because they only deal with certain types of faults. Proactive approaches increase efficiency by reducing the cost and time of operations and network resource usage by maintaining the state of executing applications and resuming operation when rescheduled.

The agent oriented approach as specified in Autonomic Computing framework deal with individual faults proactively. Agents maintain information about hardware conditions, executing process memory consumption, available resources, network conditions and component mean time to failure. Based on this information and critical states, agent can improve the reliability and efficiency of Grid services.

3.2 Autonomic Computing

Autonomic Computing can be seen as a holistic vision that enables a computing system to "deliver much more automation than the sum of its individually self-managed parts" [23]. It is considered a collection of computing resources working together to perform a specific set of functions.

3.2.1 Vision of Autonomic Computing

"It's time to design and build computing systems capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them. These Autonomic systems must anticipate needs and allow users to concentrate on what they want to accomplish rather than

figuring how to rig the computing systems to get them there. . . . It is the self-governing operation of the entire system, and not just parts of it, which delivers the ultimate benefit" [IBM Corporation, 2001] [24].

Autonomic Computing helps to address complexity by using technology to manage technology. The term *Autonomic* is derived from human biology [25]. The Autonomic nervous system monitors your heartbeat, checks your blood sugar level and keeps your body temperature close to 98.6°F without any conscious effort on your part. In much the same way, self-managing Autonomic capabilities anticipate IT system requirements and resolve problems with minimal human intervention. As a result, IT professionals can focus on tasks with higher value to the business.

3.2.2 Attributes of Autonomic Computing

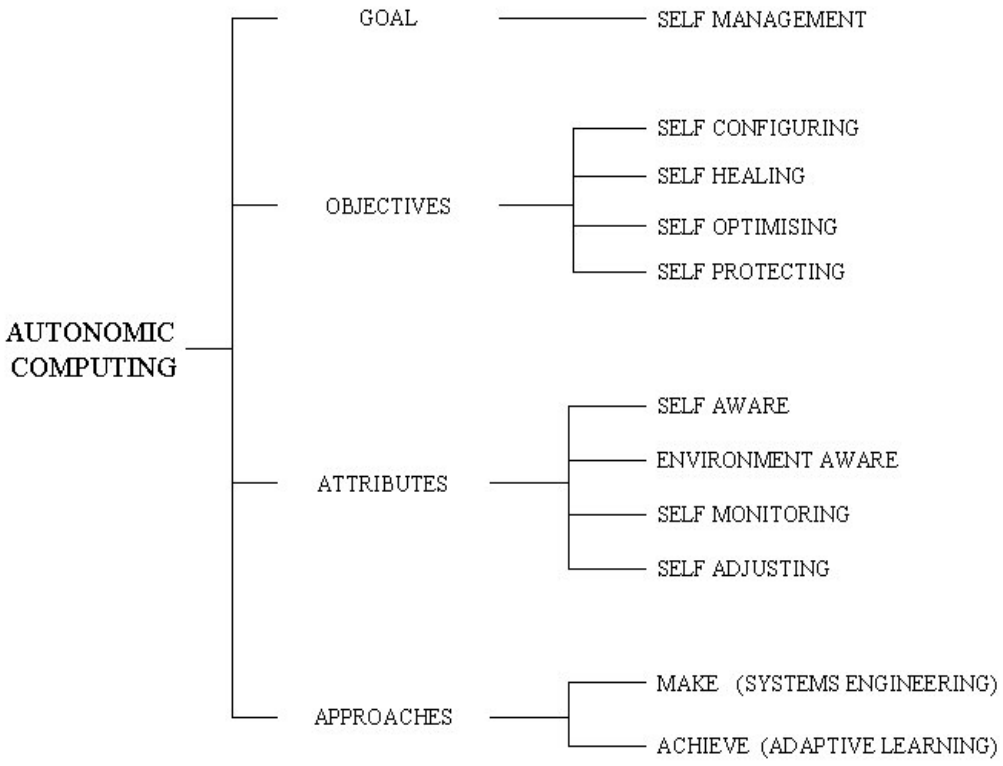


Figure 3.2 Vision of Autonomic Computing [26]

The overall goal of Autonomic Computing is the creation of self-managing systems; these are proactive, robust, adaptable and easy to use. Such objectives are achieved though *self-protecting*, *self-configuring*, *self-healing* and *self-optimizing* activities [26].

To achieve these objectives a system must be both *self-aware* and *environment-aware*, which means that it must have some concept of the current state of both itself and its operating environment.

It must then *self-monitor* to recognize any change in that state that may require modification (*self-adjusting*) to meet its overall self-managing goal. In more detail, this means a system having knowledge of its available resources, its components, their desired performance characteristics, their current status, and the status of inter-connections with other systems. The ability to operate in a heterogeneous environment requires the use of open standards to understand and communicate with other systems. In effect, Autonomic systems are proactive in their operation, hiding away much of the associated complexity from users.

3.3 Characteristics of Autonomic Computing

The following are the four main characteristics of the Autonomic Computing [27].

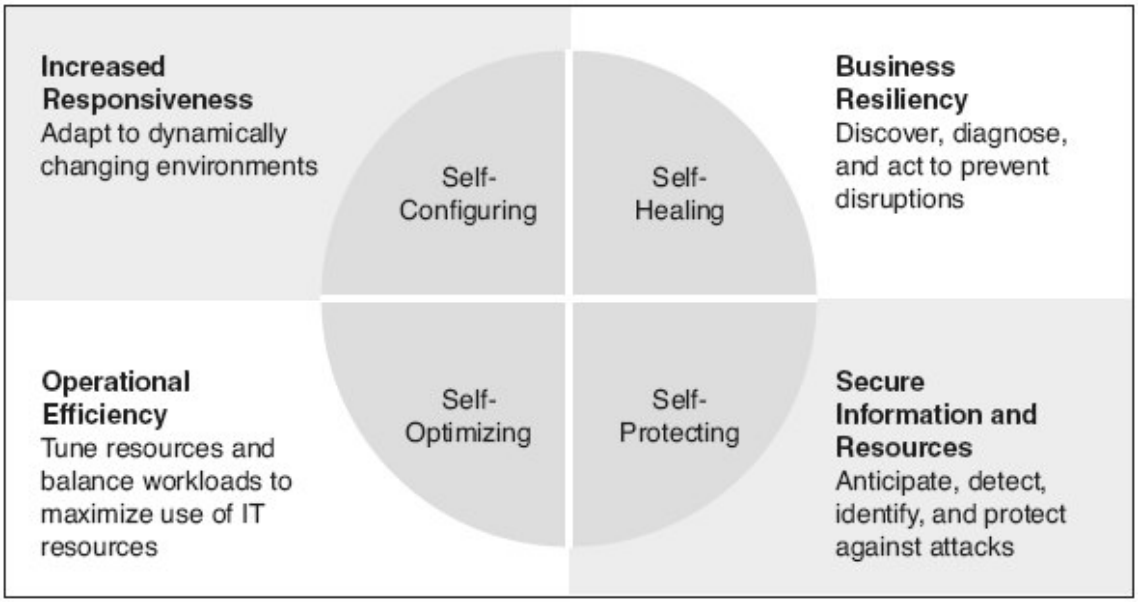


Figure 3.3 Characteristics of Autonomic Systems [27].

3.3.1 Self-Configuring

An autonomous computing system must be able to install and set up software automatically. To do so, it utilizes dynamic software configuration techniques, which means applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configurable item. Also to control changes to those characteristics, to record and report change processing and implementation status, and to verify compliance with specified service levels.

Also, downloading new versions of software and installing regular service packs are required. When working with other autonomous components, an autonomous system updates new signatures for virus protection and security levels. Self-configuration use adaptive algorithms to determine the optimum configurations.

3.3.2 Self-Optimizing

An autonomous system will never settle for the status quo. It will be constantly monitoring predefined system goals or performance levels to ensure that all systems are running at optimum levels.

Self-optimization is the key to allocate e-utility-type resources—determining when an increase in processing cycles is needed, how much in needed, where they are needed, and for how long. To be effective, autonomous self-optimization need advanced data and feedback.

3.3.3 Self-Healing

Autonomous computing systems have the ability to discover and repair potential problems to ensure that the systems run smoothly.

Self-healing systems are able to take immediate action to resolve the issue, even if further analysis is required. Rules for self-healing will need to be defined and applied.

3.3.4 Self-Protecting

Autonomous systems identify, detect, and protect valuable corporate assets from numerous threats. They maintain integrity and accuracy and are responsible for overall system security. Threats must be identified quickly and protective action taken. Autonomic system solutions address all aspects of system security at the platform, operating system, network, application, Internet, and infrastructure levels. This involves developing new cryptographic techniques and algorithms, their secure implementation, and designing secure networking protocols, operating environments, and mechanisms to monitor and maintain overall system integrity. Such security solutions need to be standardized to provide/preserve interoperability and to ensure that these techniques are used in a correct way.

To achieve this, continuous sensors feeding data to a protection center is required. A log of events is written and accessed when appropriate for audit purposes. To manage the threat levels, we might expect a tiered level. Threats can be escalated through the tiers for increasing action and priority.

3.4 Autonomic Computing Architecture

The Autonomic Computing Architecture is of two types

- Multi-Agent System [28].
- Architecture design-based, Autonomic System [29].

3.4.1 Multi Agents System

The architecture of the Multi Agent system has the following components. Each Element in the system is managed by its own Autonomic manager called agent. These agents communicate with other elements through their agents to provide the overall Autonomic behavior of the system.

- *Autonomic Element:* An Autonomic element is defined as individual system that contains resources and delivers services to humans and other systems. An Autonomic element consists of one or more managed elements coupled with a single Autonomic manager that controls and represents them [30].
- *Managed Elements:* The managed element is a controlled system component. It can be a single resource (a server, database server or router) or a collection of resources (a pool of servers, cluster or business application).

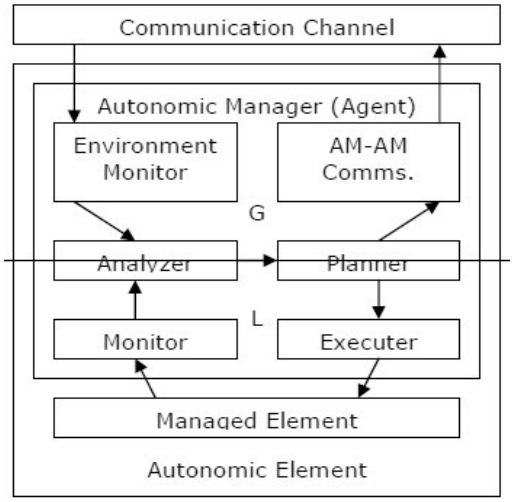


Figure 3.4 Autonomic Element [30]

The managed element is controlled through its sensors and effectors:

- The sensors provide mechanisms to collect information about the state and state transition of an element.
- The effectors are mechanisms that change the state of an element.

The combination of sensors and effectors form the manageability interface of managed element and can be available to an Autonomic manager.

- *Autonomic Manager:* The Autonomic manager is a component that implements an intelligent control loop to control the managed element. This loop can be divided into four parts (MAPE) that share knowledge [31]:
  - The *monitor part* provides the mechanisms that collect, aggregate, filter, manager and report details (metrics and topologies) collected from an element.
  - The *analyzer part* provide the mechanisms to correlate and model complex situations (for example, time-series forecasting and queuing models)
  - The *planner part* provides the mechanisms to structure the action needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.
  - The *executer part* provides the mechanisms that control the execution of a plan with considerations for on-the-fly updates

The Autonomic manager is the control part of an Autonomic element. The Interactions between different Autonomic elements are maintained with their associated Autonomic manager, and through those interactions a global Autonomic Computing architecture is established.

3.4.2 Architecture Design-Based, Autonomic System

In the architecture-based approach, the individual components are not per se Autonomic, instead the infrastructure that handles the Autonomic behavior of the system uses an architectural description model of the running system (which is not Autonomic itself) to monitor the running system, reason about it and determine appropriate adaptive actions. The adaptive infrastructure is typically clearly separated from the running system.

First of all, an architecture model is used to design the system (as is often the case with software development). In essence, an architecture model can be considered a graph of interacting components [32]. The nodes of a graph are called *components*, a general concept, and what a component actually is depends on the application. It may be desirable, however, to have a finer level of componentization. For instance, user interfaces could be considered components. The arcs in the graph are called *connectors* and they represent the interaction paths between components. Here, the granularity level of the notion of components in the model is not necessarily the same for all architecture descriptions, but is determined be the designer of the architectural model of a specific system.

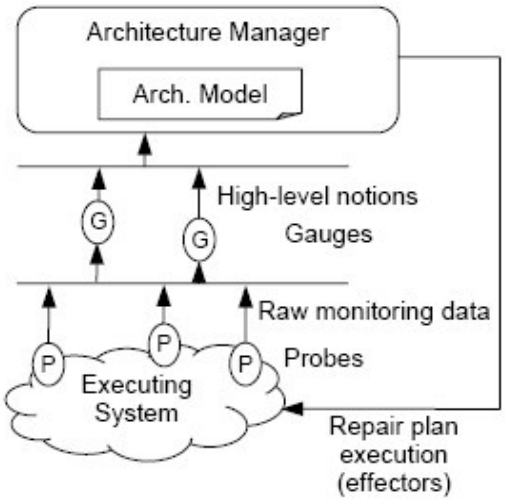


Figure 3.5 Architecture Design-Based, Autonomic Systems [32].

Many systems allow components and connectors to be annotated with a property list and constraints. These properties are updated during monitoring of the running system and the constraints on them are used to decide when an adaptation is necessary. The Autonomic infrastructure is loosely coupled with the running system. In fact, it can run on a different machine, so as not to hinder the running system. In some the code of components is augmented with checkpoints for example to allow reporting of the occurrence of specific method calls, thereby making monitoring more straightforward. Figure 3.5 shows a diagram of an architecture model-based Autonomic system illustrating the monitoring infrastructure. *Probes* can be inserted into the running system to monitor it. These probes are usually localized and deliver system-specific observations.

For example, a probe might be deployed to report the size of files that are loaded into a system, in which case the appropriate system call in the OS would be instrumented to allow this type of monitoring by a probe. The raw monitoring data provided by the probes must be aggregated and mapped to high-level notions in the architecture model. This job is performed by so-called *gauges*, which are intermediary components between the probes in the running system and the architecture manager, which controls adaptation of the system at the architecture model level [33]. Gauges may need to collect data from various probes to be able to compute high-level observations. These high-level data allow the architecture model to be updated based on the current state of the executing system. When a property in the architecture model is updated through monitoring, the architecture model is analyzed to determine whether the system is still performing adequately. If not, a repair plan is created. The repair plan

describes which components or connectors are to be removed or adjusted and which ones are to be inserted. The repair plan is created based on repair strategies that are defined in advance. For many of the architectures the adaptive strategy is closed in the future we may see the knowledge of the success of past repair plans used to determine the best strategy. This can be determined 'of-line' on another machine, however a considerable amount of bandwidth may be required for monitoring, and this can become a problem if the monitoring data travels on the same network interface as application data.

An advantage of the complete separation between Autonomic behavior and the running system is that software adaptation can be "plugged into" a pre-existing system.

3.5 Architecture of Self-Organized Autonomic Grid System

The general blueprint of an Autonomic system can be mapped on the Grid architecture to make the Grid Autonomic [34]. In the model, agents are Autonomic managers of Autonomic elements, governing several Grid services and lower-level agents. Agent governs internal affairs including service management (registration, invoking, monitoring, and exception handling) and coordination of lower-level agents, as well as external affairs for interaction and collaboration.

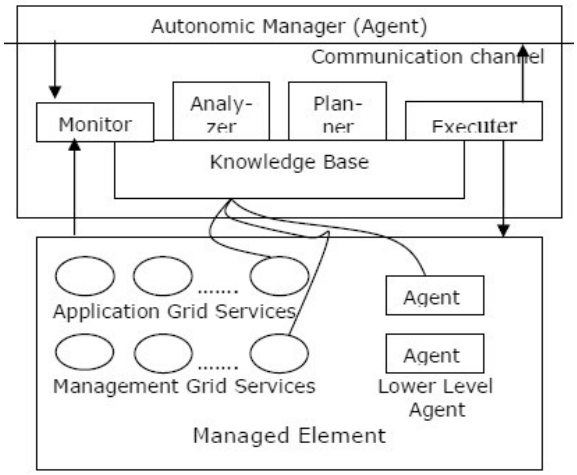


Figure 3.6 Structure of Agent-Managed Autonomic Element

There are two types of Grid services: *management services* which are Grid services used for system management works and *application services* which are those services applied for human demand [34]. Management services are the elementary function units constituting system management works. The implementation of a self-

management task can be seen as a sequence of management service executions controlled by Autonomic elements (agents). The structure of the agent managed Autonomic element is shown in figure 3.6.

- *Agent-managed Autonomic element*

From the view of Autonomic Computing, Autonomic element (AE) is composed of one Autonomic manager and several managed elements. In this model the Autonomic manager of each Autonomic element is an agent, and managed elements are application Grid services, management Grid services, and/or lower-level agents. The manager agent integrates services and lower-level agents to perform high-level functions, and undertakes management works for managed elements in their whole life cycles. External interactions, such as information exchange and cooperative work, are also carried out through agents' communication interfaces.

Autonomic element is defined as a 3-tuple: AE = (AM, ME, IR) ……. (1)

- AM – Autonomic manager, which is an agent.
- ME – managed elements, which can be application and management Grid services, and/or lower-level agents.
- IR – internal relationships between the AM and ME's.

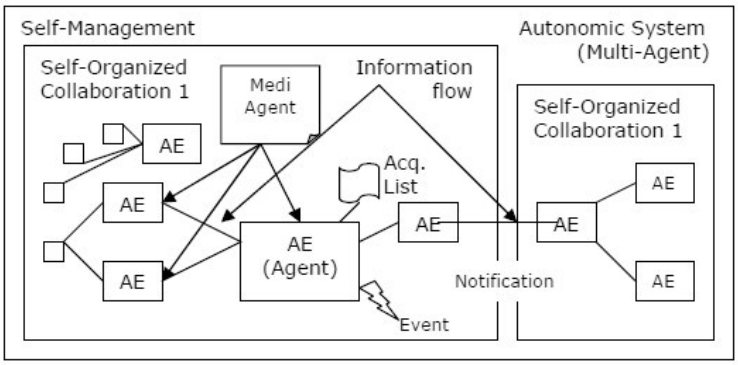


Figure 3.7 Architecture of the Self-Organized Agent-Enabling Autonomic Grid System

Composed of agent-managed Autonomic elements, Autonomic system can be seen as a multi-agent system that enables Autonomic Computing. Each agent publishes and advises its c*apability* and *interest* to other agents, including mediator-agents which mediate matching between service request and provision, and records such information of others. Capability is what an agent is able to perform, and interest is which information an agent considers important for further actions. When a system management work is needed, agents transmits useful information and task requests to others according to the capability and interest information, constituting a multi-agent

collaboration to carry out the management work. Each participant in the collaboration uses its managed elements (services, lower-level agents) to execute one or more sub-tasks, and they together perform a self management work.

The architecture of the self organized agent-enabling Autonomic Grid system (SAAGS) is shown in figure 2. SAAGS can be defined as a 3-tuple:

SAAGS = (AE, ER, SOM)……. (2)

AE – set of Autonomic elements.

ER – set of relationships between AE's.

SOM – self-organized mechanism of agent-enabling system management.

3.6 Monitoring in Autonomic Systems

Monitoring is the heart of the Fault Tolerant system.

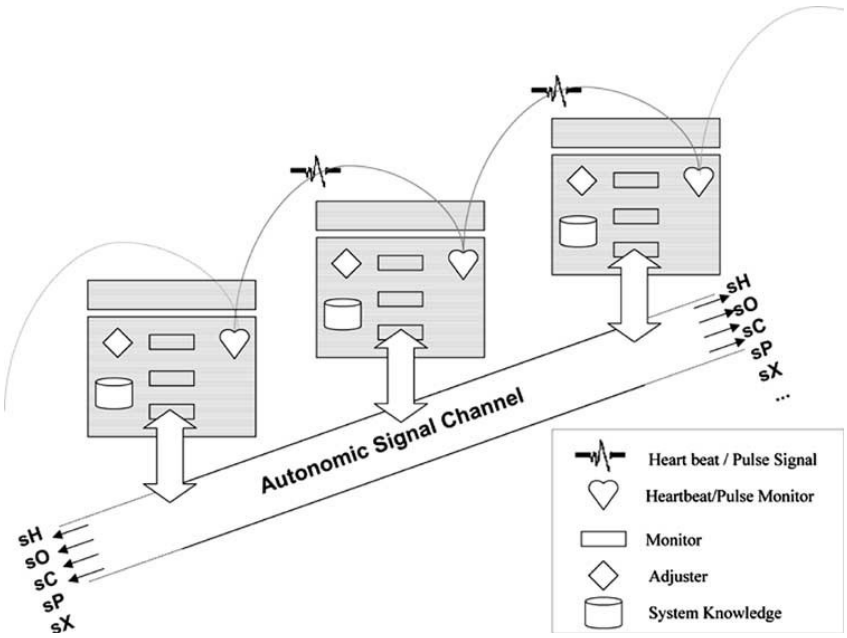


Figure 3.8 Monitoring of Autonomic Systems [35].

It has been discussed that the Autonomic manager must implement the required self-monitoring. The state of the sensors (to the managed component) are constantly being observed by the **internal monitor** and evaluated and assessed for possible action by the self- monitor [35]. A system knowledge base exists for reference containing the expected states of the system which may then be used as a means of comparison with the actual observed behavior of the system. Deviations are reported to the self adjuster for action, which may result in changes to the managed component through the effectors. The heartbeat function through registration with external elements provides a security mechanism - 'I am alive' monitoring.

Similarly, an external monitor observes the state of the environment via an Autonomic signal channel and this also may trigger internal changes. The signal channel provides linkage to other Autonomic managers.

3.6.1 Heartbeat Monitoring

The Globus OGSA (open Grid services architecture) has a health-check facility referred to as the Globus Heartbeat Monitor [36]. It is designed to detect and report processes that fail to provide a 'heartbeat'.

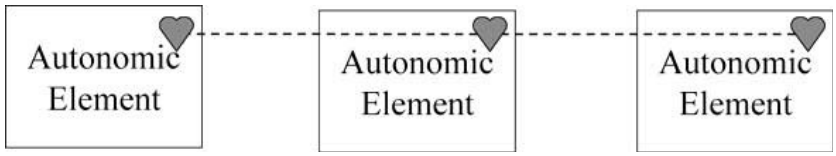


Figure 3.9 Heartbeat Monitoring [36]

Within the Globus OGSA specification the HBM consists of

- HBM Client Library,
- HBM Local Monitor and
- HBM Data Collector.

Essentially a Local Monitor runs on each host, checking and reporting on the status of the monitored processes and the actual system generating "I-am-alive" messages (heartbeats). The Data Collector receives heartbeat messages and identifies failed components from missing heartbeats. There may be one or more data collector per application.

The heartbeat monitor function ensures an element is operating. More information is required, however, to determine how well an element is performing, if it is necessary to improve its operation, consistent with the needs of Autonomic Computing.

3.6.2 Beacon Monitoring

Beacon monitoring was first used in NASA missions, particularly those to deep space [37]. In high-level concept terms, the beacon monitor is similar to the heartbeat monitor in Grid Computing, with the addition of a tone to indicate the degree of urgency involved. The various urgency levels that are present in beacon monitoring are given in table 3.1.

Table 3.1 Summary of Tone Definitions [39].

Tone	Definition
Nominal	All functions as expected no need to downlink
Interesting	Interesting - non-urgent event. Establish communication when convenient
Important	Communication need to take place within timeframe or else state could deteriorate
Urgent	A critical component has failed. Cannot recover autonomously and intervention is necessary immediately
No_Tone	Beacon mode & is not operating

3.6.3 Pulse Monitoring

A hybrid approach for the Autonomic environment [39] is to use the urgency concept of the beacon monitor to turn the heartbeat monitor into a **pulse monitor-**so instead of just checking the presence of a beat; the rate is also measured as shown in Figure 3.10.

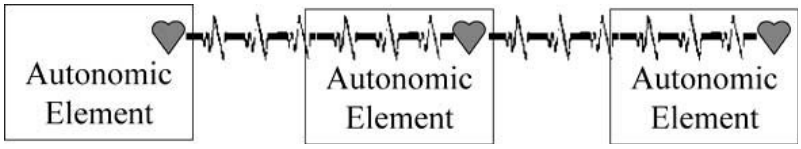


Figure 3.10 Pulse Monitoring [39].

The concept of the pulse monitor is based on extending the HBM construct. The HBM itself provides a means to ensure a vital process may be safeguarded. The lack of a heartbeat will alert the designated remote HBM that the process has died (or indeed the communications themselves have failed). This relative instant alert to the fact a process is no longer functioning enables immediate actions such as restarting the process and as such minimizing disruption.

Yet, vital as it is, essentially the HBM only informs if a process is alive or dead (assuming communications are working) - not the processes actual health or state of being. Taking the biological analogy the rate of the heartbeat indicates the current conditions within which the biological ‘system’ is operating and determines strategies for ‘components’ within the system.

An important point to note from the HBM, and also from the Beacon Monitor, is the minimization of data sent - essentially only a ‘signal’ is transmitted. Any move

towards sending more information must not compromise this reflex reaction. **As** such the tone or the beat must contain within it the urgency level.

This effectively provides a reflex reaction within the Autonomic Grid environment and adds a duel approach, sharing responsibility for environment monitoring and indicating increasing urgency levels.

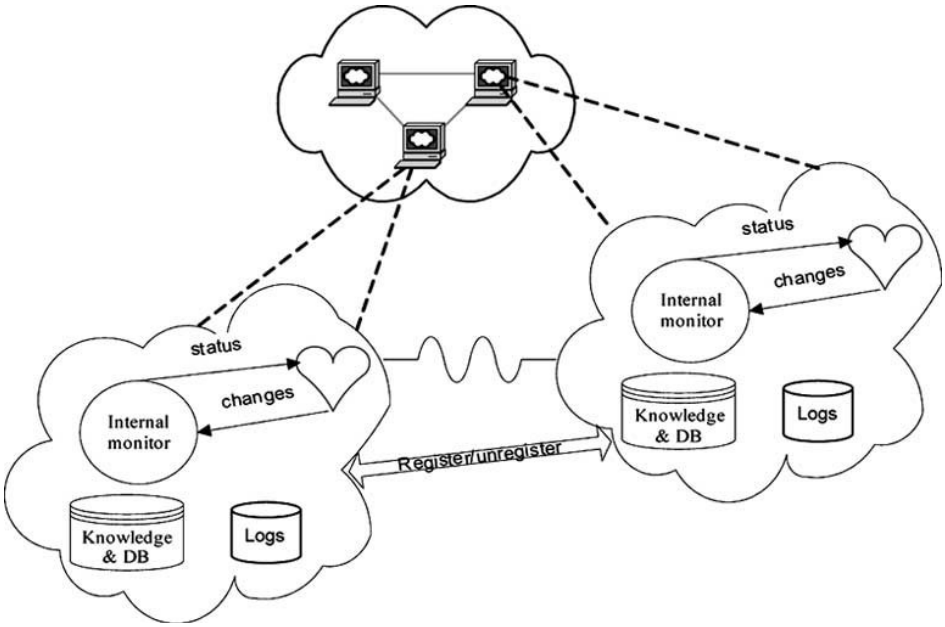


Figure 3.11 Pulse Monitoring Functionality [40]

The pulse monitor's functionality still includes the standard HBM function - the present or absence of a beat indicates the process is still alive. Yet the pulse monitor must also contain the pulse levels and rules concerning the changing between levels, the ability to resolve conflicts in perceived levels to enable the extended health check functionality. The local internal environment must be alerted upon receiving a confirmed change in levels from the external environment while also being able to recommend a change due to changing circumstances it has detected through its own self-monitoring that will have an effect on the external environment.

3.7 Future of Monitoring- Towards Affect and Emotion

Looking further into the future, the Grid architecture would be more autonomous if only it had 'feelings' or emotions, a trait strongly suggested by some psychologists and AI researchers that is essential for intelligent behavior. This desirable feature is described in biology in organisms as the system of affect and emotion which evaluates the environment with respect to how it would affect the organism/system

under consideration. It usually results in some sort of overall *feeling* of whether the situation is positive/negative, safe/dangerous etc.

In establishing the Autonomic Grid, it is important the system of affect and emotion be taken into consideration and how such a feature may be modeled and incorporated. Some preliminary work has been carried out by Norman et al. [41] who introduce a system of affect as running in parallel with a system of cognition. Cognition is defined by [42] as the system which *makes sense of the world* by understanding and reflecting on things in the world. Affect, on the other hand, is defined to be more focused on *evaluation* of the events in the world and their value with respect to the organism/system. In other words the affect system gives a warning about possible danger and may lead to the body becoming more alert and ready for danger while potentially changing the cognitive strategy.

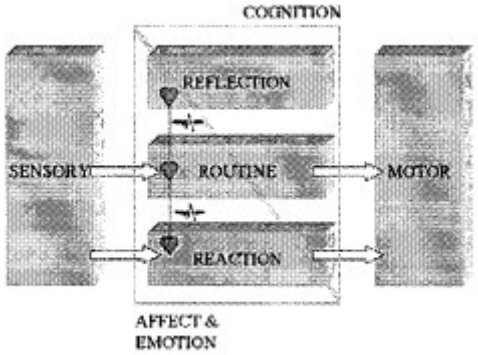


Figure 3.12 Cognition System

The three levels of Norman et al's model arc named as reaction (lowest level), routine and reflection:

- Reaction - lowest level where no learning occurs but immediate response to state information coming from sensory systems.
- Routine - middle level where largely routine evaluation and planning behaviors take place. It receives input from sensors as well as from the reaction level and reflection level. This levels assessment results in three dimension affect and emotion values: positive affect, negative affect and (energetic) arousal.
- Reflection - top level receives no sensory input or has no motor output, it receives input from below. Reflection is a meta-process, where the mind deliberates about itself. Essentially operations at this level look at the systems representations of its experiences, its current behavior, its current environment etc.

Fault tolerance features can be incorporated into the Grid with the implementation of Self-Healing part of Autonomic Computing. By implementing the generic Autonomic Computing framework we can achieve the self- healing characteristic of Grid and in other terms Grid can be made fault tolerant. This chapter described in detail, the fault tolerance in Grid and the Autonomic System, its characteristics, its Architecture and its need. The architecture of self managed Grid has been depicted and discussed. The Monitoring part of the Autonomic System has been discussed in detail, with various other Health check mechanisms. In the next chapter, the problems in the existing Grid have been discussed.

Chapter 4 Problem Formulation

This chapter gives a detailed description of the problem found in existing approaches of Fault Tolerance at the cluster level in a Grid environment and the methodology used to solve this problem.

4.1 Limitations at the Cluster Level

The clusters are often used as computing nodes in global Grid environments at a conceptually higher level. The vastly grown requirements on the availability, however, clearly identify the neuralgic shortcomings in the current state-of-the-art of the cluster technology, namely their poor fault tolerance and high maintenance overhead. Moreover, clusters are often incrementally installed or upgraded over time, typically resulting in heterogeneous hardware and software and thereby making it difficult to manage them in a consistent way.

Here, the mentioned problems become even more critical, because single clusters should (ideally) function autonomously without human intervention. The automated operation is not only necessary for reducing human administration effort, but also to limit possible sources of errors and to reduce service downtimes.

Fault tolerance at Fabric layer is generally not provided by the Grid middleware. The problem with *self-management* of all hardware and software components includes

- Ensuring the functional integrity of hardware and software components
- Switching off faulty services
- Removing hosts from the active set of machines.

A Grid Computing center is build of many services such as a file server, a DNS server, a Grid access node, a batch master, a batch worker, etc. The relationship of services to machines (hardware) is typically *N* to *M*:

- some services exist only once (e.g. DNS server),
- multiple machines host a single specific service only (e.g. file server and batch worker) and some machines host multiple services (e.g. Grid access and batch master).

The *self-management* of the cluster must ensure that the services are running and functioning correctly, e.g. by performing basic tests. Also, the hardware must be monitored, e.g. to simplify the identification of faults.

4.2 Objective

Fault Tolerance is required to be achieved in order to make Self-Healing possible in the Grid environment, which is one of the main characteristic of the Autonomic Computing. The main objective of the thesis work is to incorporate the proactive fault tolerance approach in the Fabric layer of the Grid (Cluster level).

4.3 Methodology

As discussed, the limitations in the cluster level of the Grid are the main hindrances in achieving the fault tolerance in the Grid.

As a solution, the main agent running on the system works as a monitor, looking for, if any fault has occurred. On detection of the fault, the main agent provides the recovery and necessary fault tolerance. This prototype has been designed to incorporate the proactive fault tolerance approach in the cluster level of the Grid.

In the next chapter, the design of the prototype has been discussed and the implementation details are given in successive chapters.

Chapter 5 Proposed Solution

This chapter describes the proposed Architecture to incorporate proactive Fault tolerance approach into the Grid. The existing Architecture for the self-healing at the cluster level (Fabric) has been described. Based on this architecture, the design of the cluster prototype has been defined.

5.1 Architecture for Incorporating Fault Tolerance Features at Cluster Level (Fabric Layer) of Grid

This section discusses the existing architecture of the cluster in the fabric layer of the Grid environment. At a conceptually finer level, clusters are often used as computing nodes in global Grid environments. The problems of management of Faults in grid are even more critical, because single clusters should (ideally) function autonomously without human intervention. In this work, a system for the coordinated, autonomic management (self-healing) of multiple clusters in a compute center has been discussed. The clusters are integrated into the grid making the grid Fault tolerant. This consist of two components [43]

- Monitoring part to monitor the elements of the clusters
- Fault detection and recovery part for detecting the faults and automatically recovering from them making the cluster Fault tolerant.

5.1.1 Architecture of the Monitoring System

Monitoring data about many different components must be gathered and stored to facilitate the self-healing of fabric services.

The monitoring system consists of four components as shown in Figure 5.1.

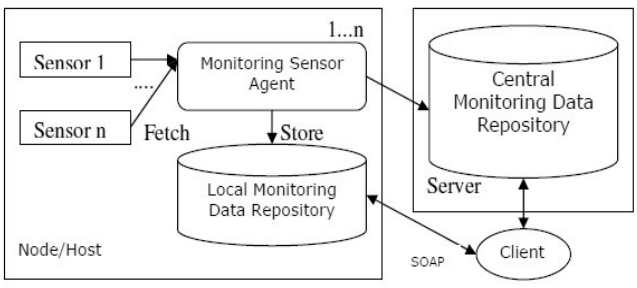


Figure 5.1 The Monitoring System Consisting of Sensors, a Sensor controller, a Local and a Central Monitoring Data Repository [44].

1) Sensors: On each machine different *sensors* periodically collect data.

2) Monitoring Sensor Agent (MSA): All sensors are controlled by the MSA, which receives data samples from the sensors

3) Local Monitoring Data Repository: Collected data by the MSA are stored in Local monitoring data repository.

4) Central Monitoring Data Repository: The MSAs also forward the data samples to a central repository.

The Monitoring system provides the overall functionality as given below [44].

- *Data Sampling:* A configurable Monitoring Sensor Agent runs on all monitored hosts. The MSA is responsible for calling the plug-in sensors to sample the configured metrics. The system provides sensors for common performance and exception monitoring. The sampling frequency and the minimal change percentage required for a sample to be sent (smoothing) can be configured as per metric. The local monitoring data repository (cache) is available for local consumers of monitoring data. This is useful for allowing local fault tolerance correlation engines and may be used to resend data to the central repository in case of sending data has failed.
- *Data Access:* Data may be accessed through the repository API. The API does not distinguish different data types (must be handled by the client of the API). The API provides methods to insert samples into a repository, to query samples and to subscribe for change notification. The result of queries may contain one to many samples.
- *Data Transport:* The transport of monitoring data from the monitored hosts to the central repository is also pluggable. Implementations for both UDP and TCP (prototypic) exist.
- *Data Storing:* The central measurement repository server uses the repository API to plug-in any database system as backend. So far, backend for flat files (same as for the local repository), Oracle, and ODBC have been developed

5.1.2 Architecture of the Fault-Tolerance System

The aim of the *Fault Detection and Recovery* system (FDR) [45] is to provide automatic error detection and correction, i.e. self-healing of a fabric. In a large cluster or fabric one faulty node can cause serious problems for the whole grid. For example, a broken DNS server or a broken gateway may disconnect a whole fabric from the

grid. Another problem may be that a normal computing node may cause a long delay in the analysis job or may even cause the total failure of a job. For the described scenario, the Fault Detection and Recovery (FDR) system must cope with automatically running tasks that are not commonly found in today's cluster management systems:

- Automatic error prevention, e.g. to prevent hardware damage caused by overheating,
- Automatic error correction, e.g. by restarting crashed services,
- Schedule repair tasks that would interfere with running jobs, e.g. freeing disk space on file servers.

The FDR system is rule based. Each rule compares data retrieved from the monitoring system against configured limits. If the condition of a rule holds, the specified action is performed. By following a hierarchical approach, the FDR system supports handling faults both locally on each machine and remotely from specific machines dedicated to maintain the correct functioning of the diverse services in a Grid computing center.

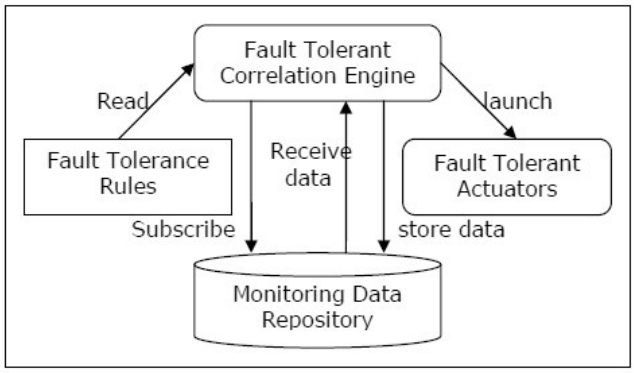


Figure 5.2 Components of the Fault-Tolerance System [46]

The components of the FDR system are shown in figure 5.2. These components are [46]:

1) Fault Tolerance Correlation Engine:

The Fault Tolerance Correlation Engine (FTCE) is the active correlation engine. The FTCE runs as a daemon process on all hosts and is implemented to be robust to most system component failures. The FTCE processes observe one or several metrics stored in the MR to determine if something has gone wrong or is on its way to go wrong on the system. If so, it determines what recovery actions are needed, and launches the corresponding actions. Its output metric values contain a boolean flag that reflects if any fault tolerance actuators were launched, and if so, the identifiers of

the actuators and their return status. The FTCE processing for a given metric is triggered either through a periodic sampling request from the MSA or through the metric subscription/notification mechanism provided by the monitoring repository.

2) Fault Tolerance Actuators:

A Fault Tolerance Actuator (FTA) is an implementation of the FaultToleranceActuator interface that executes automatic recovery actions. A given FTA implementation can thus be used for several similar recovery actions, e.g. a single "daemon restart". Another example for an FTA is a service restarter, i.e. by calling the restart method of the installed software packages.

3) Fault Tolerance Rules:

A Fault Tolerance Rule (FTR) contains all necessary information about the controlled values on the nodes and the actuators that should be started if a value runs out of its defined limit. The interface for the actuator is as simple as possible: it may be a shell or an executable.

5.2 System Design

The implementation of a Fault Tolerant cluster involves the efficient management of redundancy inherent in a cluster to eliminate single point of failure. The main aim of dissertation is to implement a prototype. Home Server, Main Server & Client nodes are the personal computers, which are connected to the network through standard Network Interface Card (NIC) as shown in Figure 5.3.

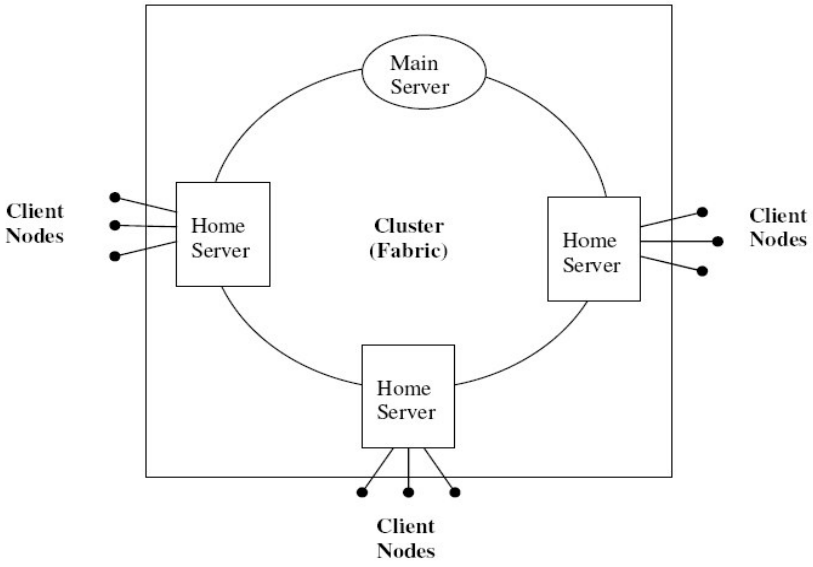


Figure 5.3 Cluster in Fabric Layer

5.2.1 The Cluster Design

In this work, the Cluster is representing the Fabric layer in the Grid, which manages all types of interaction between MainServerManager and HomeServerManager and all complex mechanisms present in the Cluster are hidden from Client nodes. New services such as Files can be easily added to the Home Server as the Cluster separates service specific logic from service availability logic. The three main components of the Cluster are: MainServerManager, HomeServerManager and ClientManager as shown in Figure 5.4.

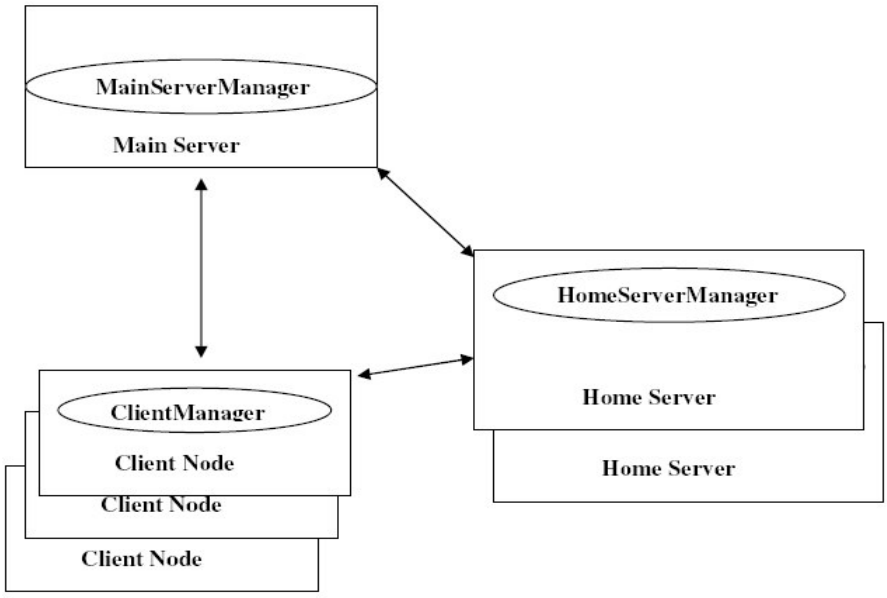


Figure 5.4 Components of the Cluster

1) MainServerManager

MainServerManager is the central controller for the fault tolerant cluster. It is the main component of the Cluster responsible for fault tolerance. The major functions of MainServerManager are:

- Checks Home Server Availability: It keeps a record of the active Home servers which are present in the system for fulfilling the client requests.
- Handles Faults: If a client is unable to establish a connection with its Home Server, the MainServerManager provides the client with another IP Address corresponding to an active server which contains the file requested by the client. In this way the request of the client is fulfilled and hence the system becomes Fault Tolerant.

2) HomeServerManager

The HomeServerManager module controls the interaction between a Home Server and the Clients present in its Domain.

It also fulfills the valid request of a client node by making the required file available to it. In order to support the fault tolerance concept each Home Server must register itself to the Main Server and HomeServerManager also implements this functionality.

3) ClientManager

ClientManager allows a user to avail the services so provided by Home Server only after checking his authenticity. User can forward his request for a file to the Home Server via a GUI maintained by ClientManager.

5.2.2 Handling of File Access Request by the Cluster

The following operations are carried out:

1) Normal Operation

When the Home Server of a domain to which Client X belongs is active.

- A client issues a file access request to the Home Server of its domain and the Home Server is active then file will be successfully communicated to the client.

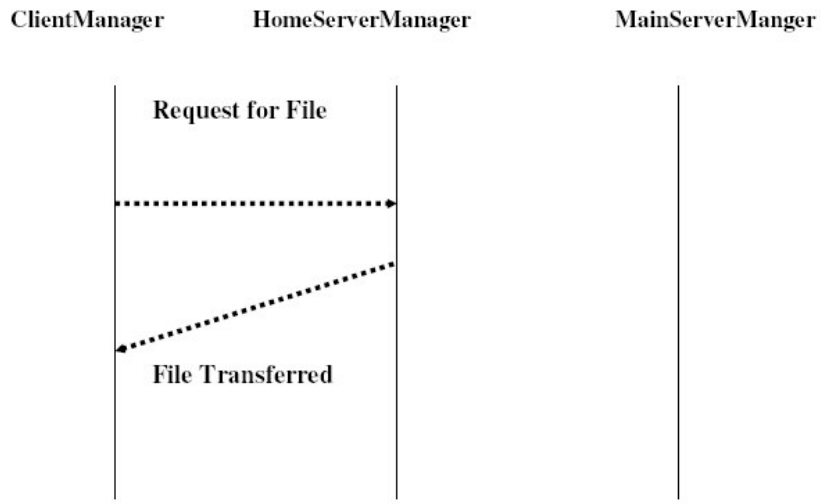


Figure 5.5 Normal File Access Operation between Client node and Home Server

*2) Operation involving Fault Tolerant approach***:**

When Home Server of a domain to which Client X belongs is not Active then following steps are managed by Cluster:

- ClientManager running on a client node detects a connection failure with its Home Server, it send a probe message to the Main Server.

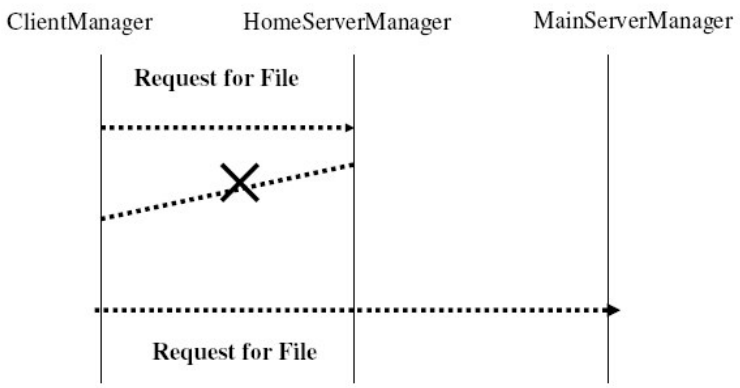


Figure 5.6 Connection Failure between Client node and Home Server

- On receiving this message MainServerManager will start searching its Address directory to find the existence of any other active Home Server. If any active Home Server is available then MainServerManager will send its IP address to the required client node, otherwise appropriate message is communicated to it.

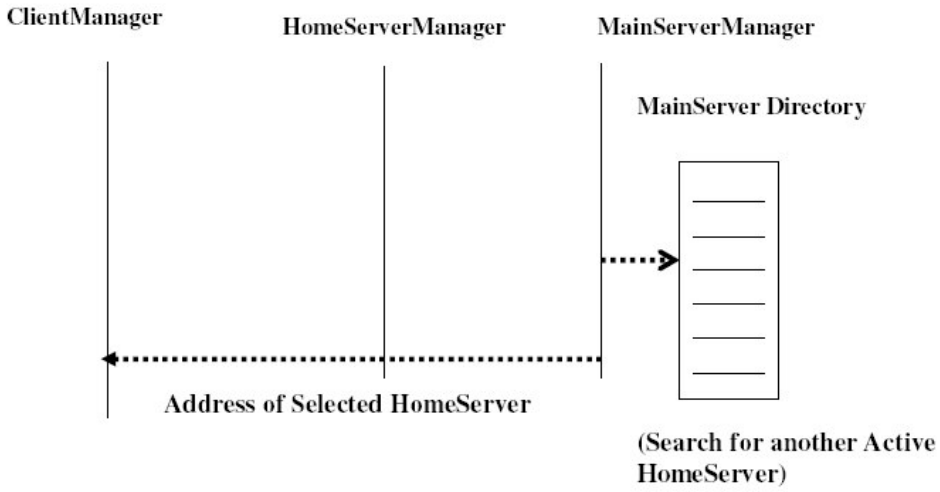


Figure 5.7 Search for Active HomeServer by MainServerManager

- On receiving the address of backup Home Server ,ClientManager will now establish connection with it to fulfill its file related request.

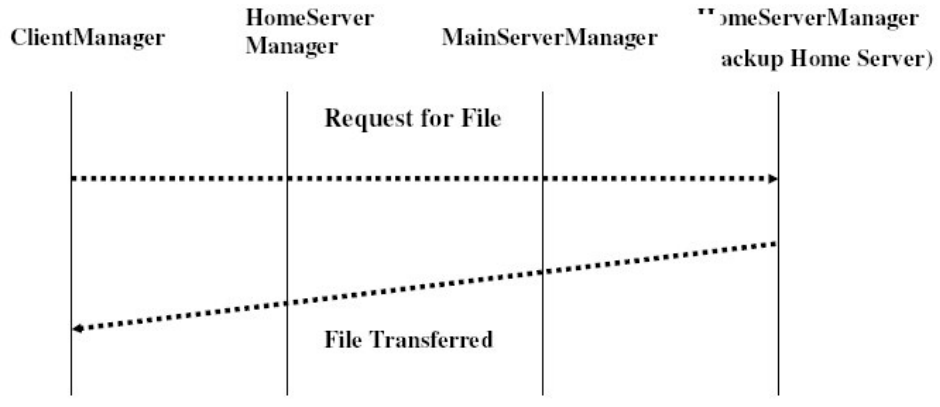
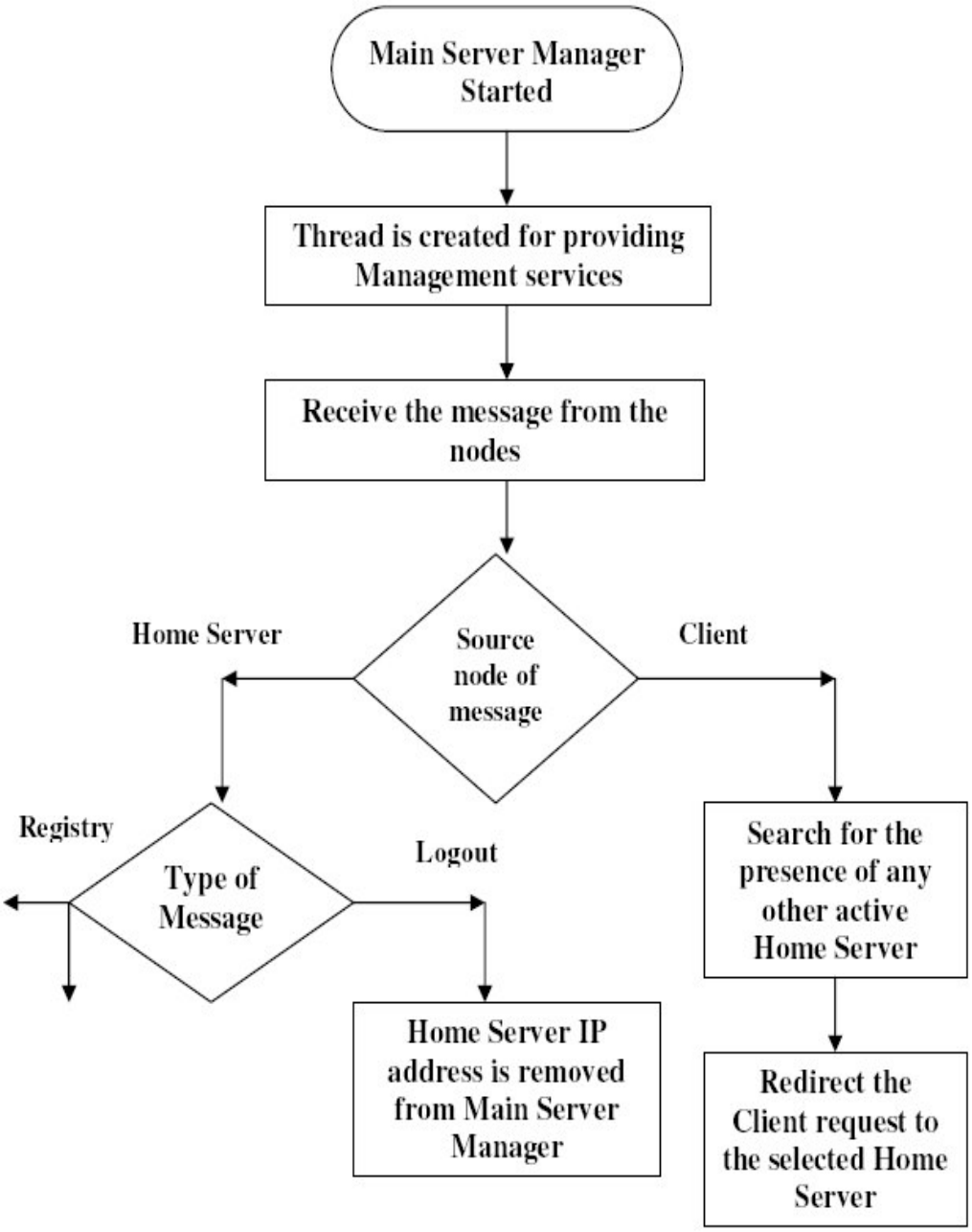


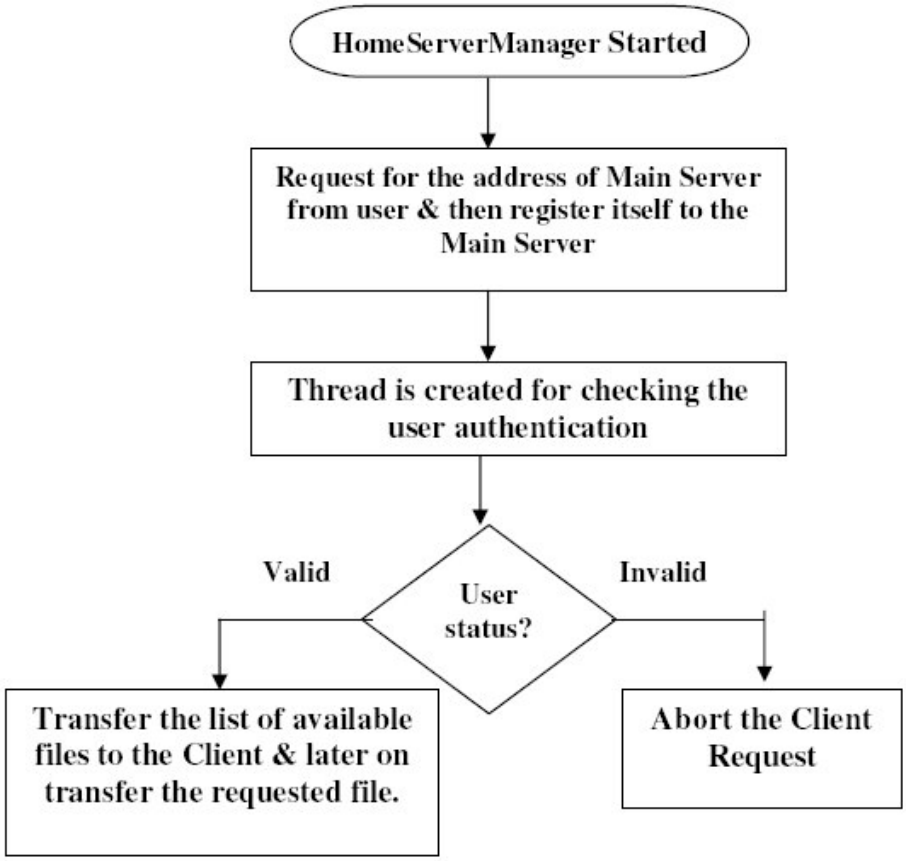
Figure 5.8 Availability of required service to a client node by Backup HomeServer

5.2.3 Flow Diagram of the Cluster Prototype

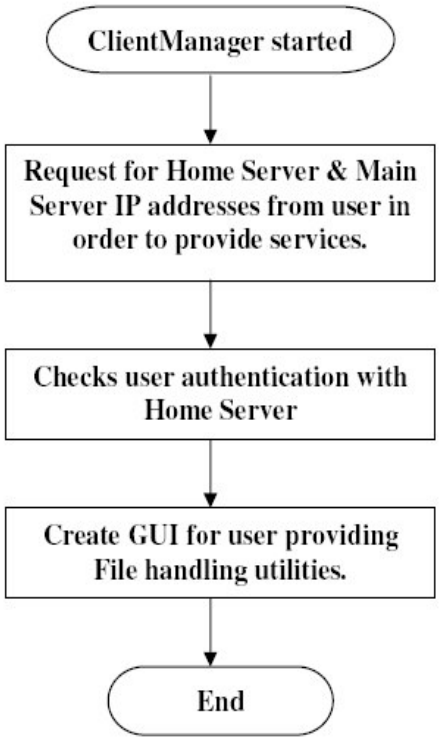
1) Flow graph for MainServerManager



2) Flow graph for HomeServerManager



3) Flow graph for ClientManager



In this chapter the Architecture for incorporating Fault tolerance features at Cluster level (Fabric Layer) have been discussed. The detailed description of the components of the architecture: Monitoring System and Fault Detection and Recovery System has been given. The design detail of the system prototype has been given.

In next chapter the implementation detail of the prototype would be discussed.

This chapter gives the detailed description of the implementation of the Cluster prototype.

6.1 Components of the Cluster Implementation

The three components MainServerManager, HomeServerManager and ClientManager have been implemented in the following way.

6.1.1 MainServerManager

The MainServerManager acts as a central server in the fault tolerant system. This forms a key part of the system which is responsible for fault tolerance. It has two main jobs

- Checks Home Server Availability: It keeps a record of the active servers which are present in the system for fulfilling the client requests.
- Handles Faults: In case of a client is unable to establish a connection with its home server, the main server provides the client with another IP Address corresponding to a active server which contains the file requested by the client.

In this way the request of the client is fulfilled and hence the system becomes Fault Tolerant.

Implementation: Class MainServerManager stores the IP Address of the Servers in a Vector. This Vector has the IP Address added in a string format. When a new server Logs in the IP corresponding to that server is pushed in the vector and similarly on a Log out, the IP is removed from the vector. In case when a client establishes a connection with the main server, it writes the IP Address of the active server to the client buffer.

6.1.2 HomeServerManager

The server is generally referred to as the Home Server. It contains all the predefined files needed to process the client's request. All the file's needed will be present in a folder named as the *Service Directory***.**

The server has the following jobs:

- *Registration with the MainServer*: Every new server when started will have to register with the MainServer. In this way the IP Address of the Home Server will be retained by the MainServer until it is active.

- *Fulfilling Client's Request*: Another main task of the server is to fulfill the requests of the client by providing them the files needed.

Implementation: Class HomeServerManager inherits properties from the predefined Thread class. It has an object of the ServerSocket class which listens for the client requests on the port no: 2000. In the run method of the thread another object of the class ThreadedServer is created. The login and the logout of the server has been defined in the main().

Class ThreadedServer is a thread which handles the following:

- *Client Authentication*: The client is verified by checking the User name and Password from the database.
- *File Transfer*: Every authenticated client will be allowed to download a file from the server.

6.1.3 ClientManager

The ClientManager can execute on any node in the network. The client has to enter the address of the home server. If the server is active the connection is established. In case the server is inactive, the client establishes a connection with the main server.

Implementation: Class ClientManager defines the constructors depending on which the appropriate frame is called.

The two frames are:

- User Authentication Frame
- File Upload Frame

The actionEvent( ) handles various button events. In case any server is not present a message "No Server Available" will be displayed.

6.2 Result and Discussions

The Results are the following screenshots with the description and the observations are as following

6.2.1 Output Screenshots

- This is the console window when you compile and execute the MainServerManager.java file

Figure 6.1 Execution of MainServerManager

- On executing HomeServerManager.java file the following dialog box appears asking for the main server IP Address for its registration with MainServerManager

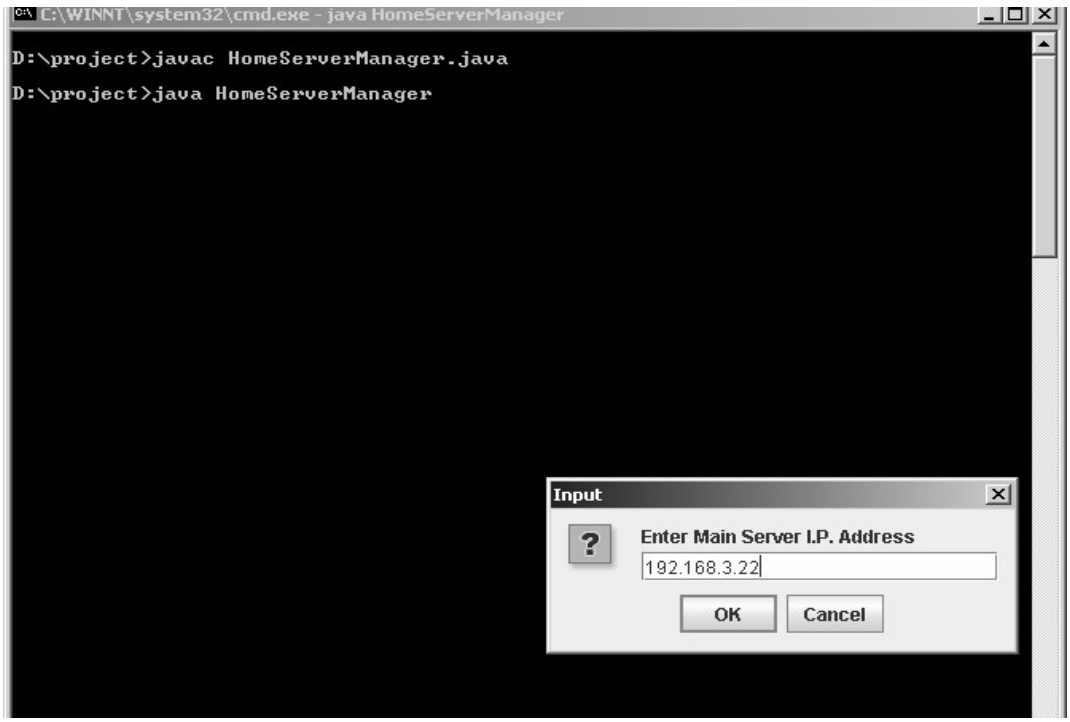


Figure 6.2 Execution of HomeServerManager

- This output appears on console of system where MainServerManager is running showing that the HomeServerManager has been added.

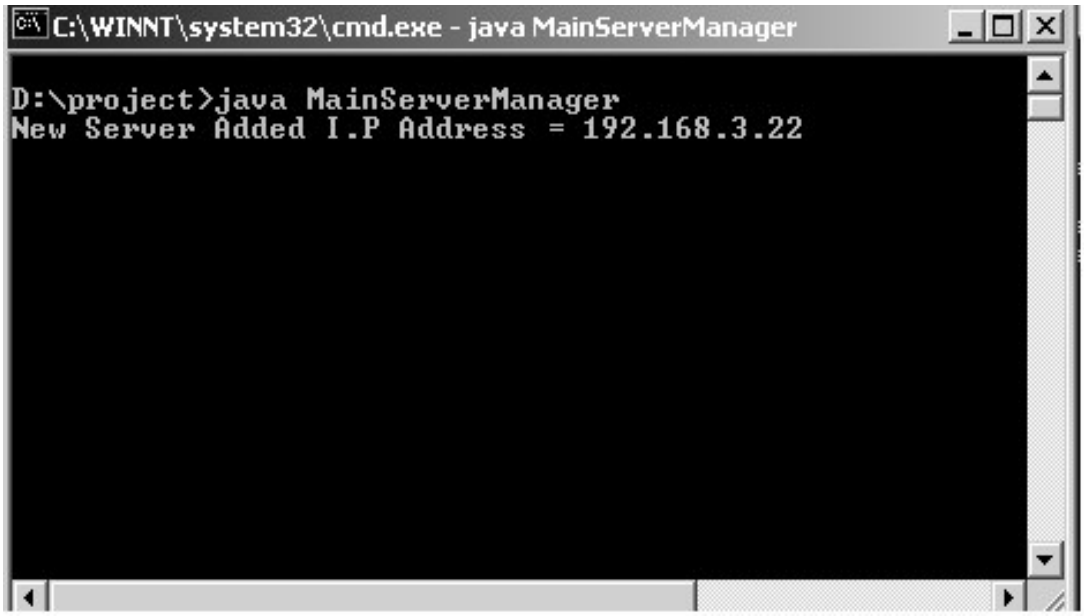


Figure 6.3 Registration of HomeServer to MainServer

- When the HomeServerManager running on different system gets registered with MainServerManager on other system, the following output appears on HomeServerManager.
- Message is displayed "Press Any Key to Logout" on console. If any key is pressed the HomeServerManager will stop its execution

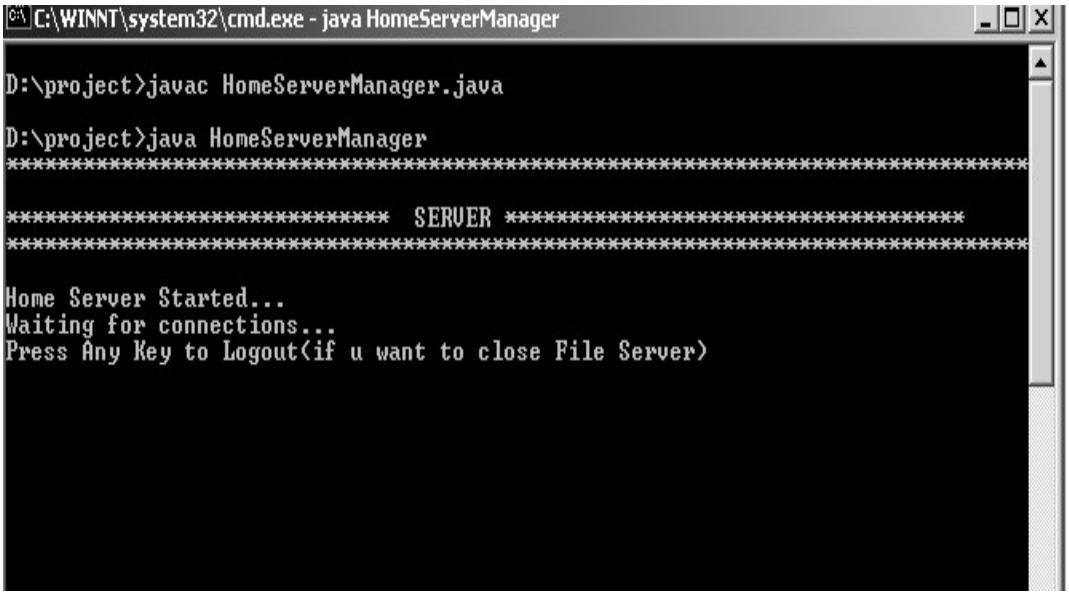


Figure 6.4 Wait Event of HomeServer

- On pressing the key, the HomeServerManager will stop running

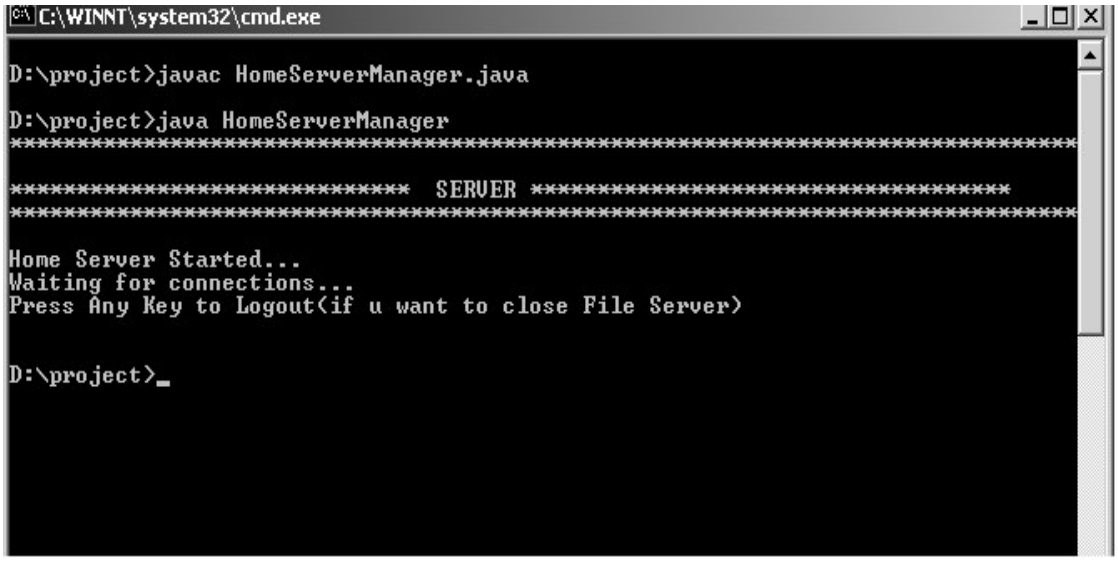


Figure 6.5 Logout of HomeServer

- On executing the ClientManager the given dialog box appear asking for the HomeServerManager IP Address. The User will enter IP Address of the HomeServerManager

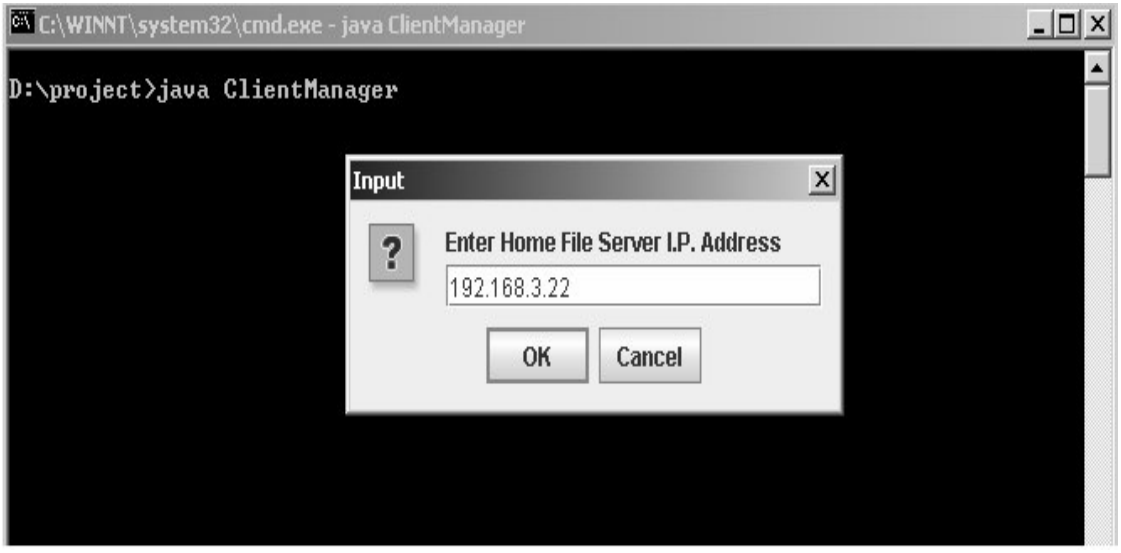


Figure 6.6 Retrieval of HomeServer Address from User by ClientManager

- Next dialog box appear asking for the MainServerManager IP Address

Figure 6.7 Retrieval of MainServer Address from User by ClientManager

- After entering the above information the AUTHENTICATION window appears on ClientManager system.
- It asks for USERNAME and PASSWORD for the authentication of the user.

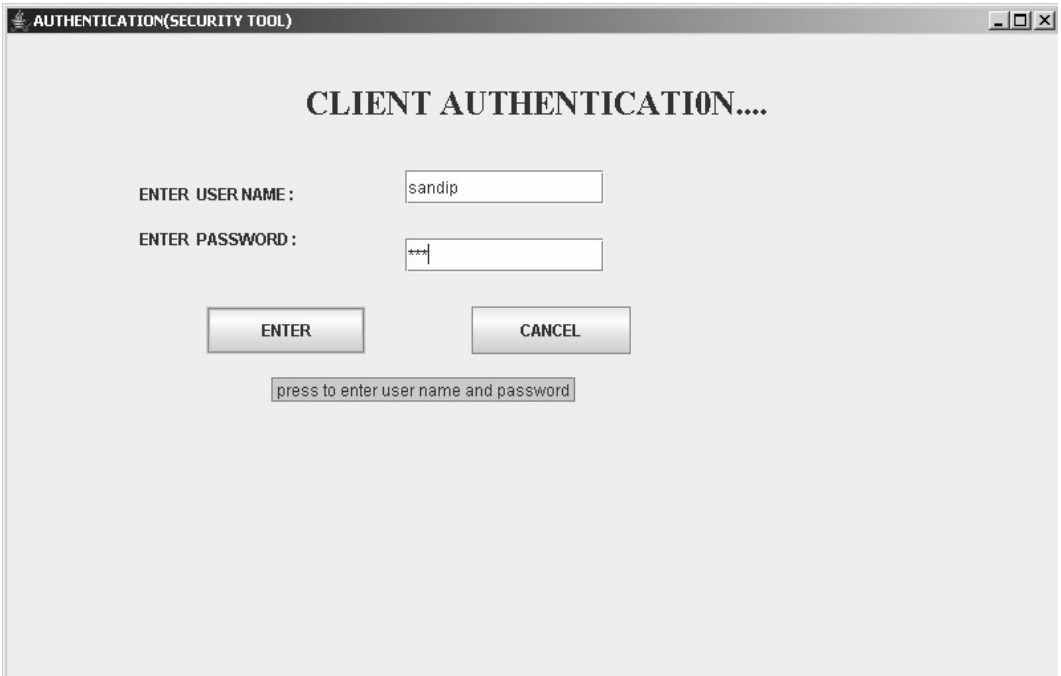


Figure 6.8 Authentication Form for User Generated by ClientManager

- On entering wrong USERNAME or PASSWORD, The dialog box appears depicting incorrect username or password and will abort the operation.

Figure 6.9 Aborting of Invalid User

- On entering wrong USERNAME or PASSWORD, The dialog box appear depicting "welcome to system" and user is authenticated

Figure 6.10 Acknowledgement for Valid User

- After authentication of user, the following console out will appear on system running HomeServerManager showing the Services available for valid user

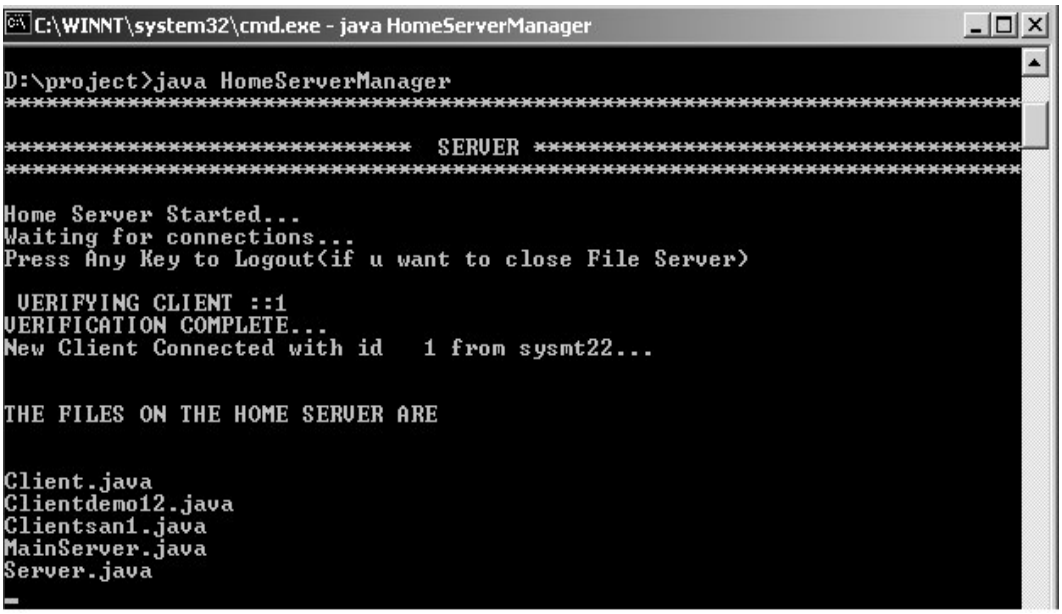


Figure 6.11 List of Services Available for Valid User

- On system running Client Manager the following Frame will Appear showing the files available on the Home Server Manager on right hand side.
- In the input box with label "enter file-name", the user will enter the file name which user wants to download.

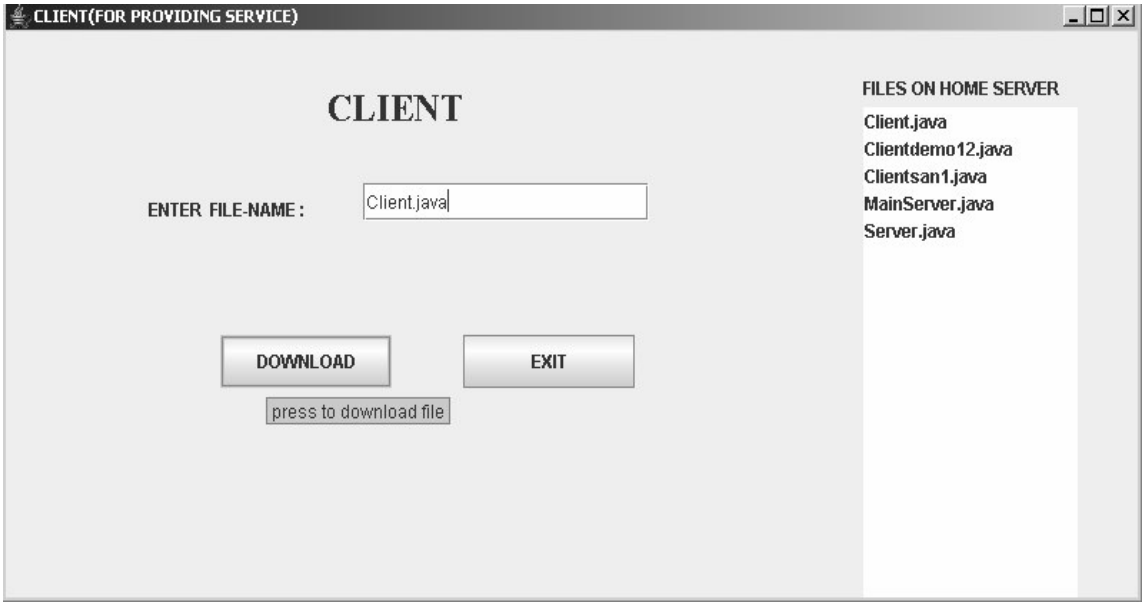


Figure 6.12 File Download Form

- When user will enter the file name in the CLIENT frame on ClientManager and Press "DOWNLOAD", the following console output will appear on HomeServerManager showing the request made by the client on ClientManager.

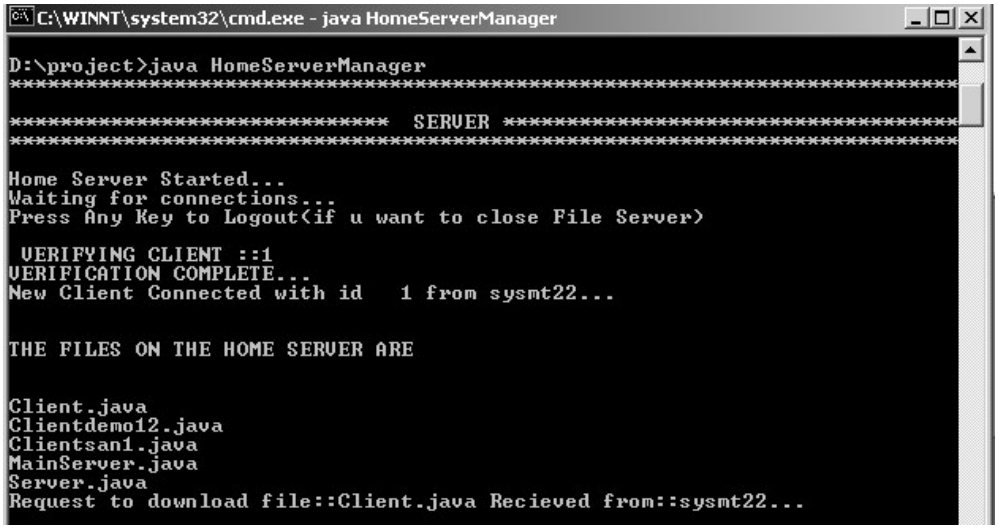


Figure 6.13 Handling of File Request by Home Server so Generated by User

- On successful completion of file download the following dialog box will appear, depicting the successful download.
- Exit button is there to exit from the CLIENT form.

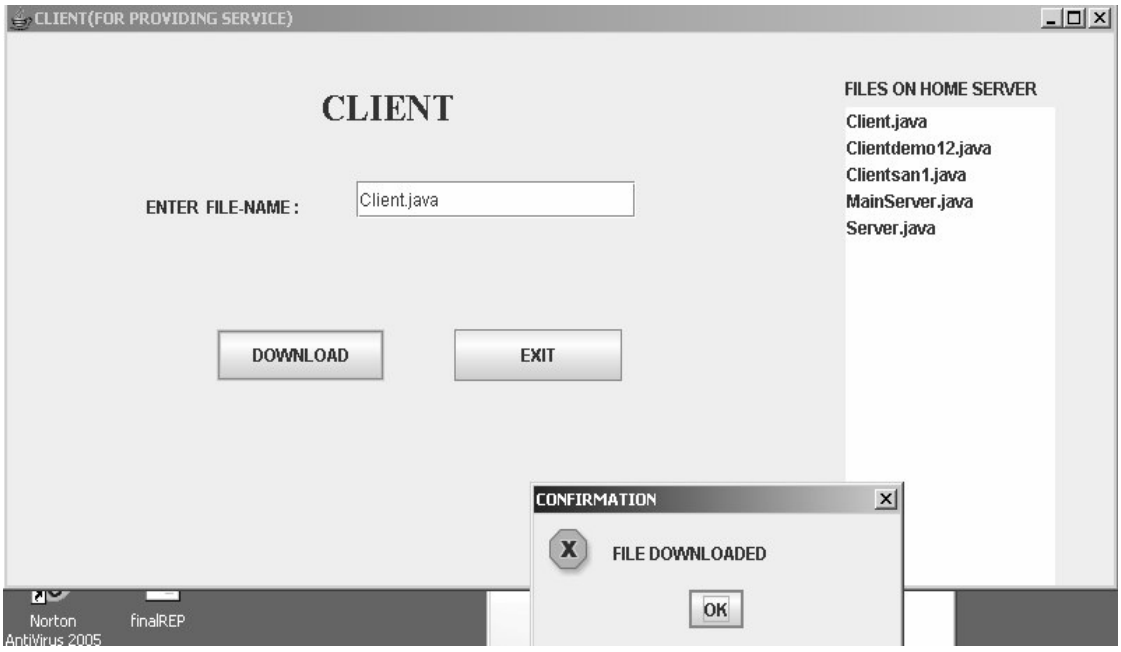


Figure 6.14 Successful Completion of File Download Operation

- On the Client Manager console widow following output will appear showing the total time elapsed to download the file.

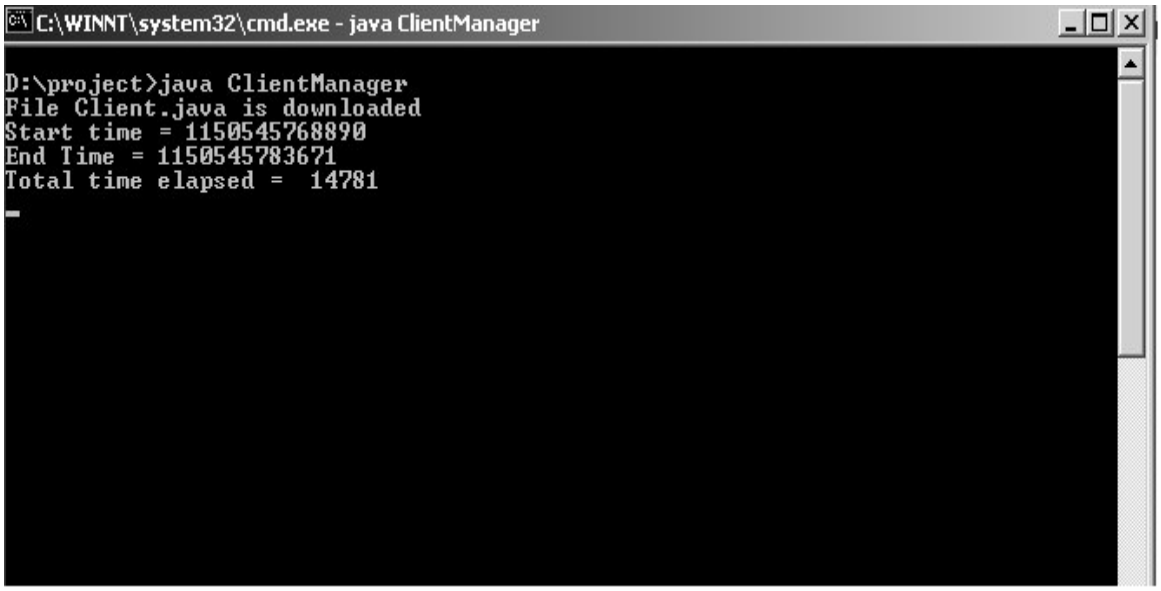


Figure 6.15 File Download Time in a Normal Mode

- If no Active server is available the following dialog box will appear on Client Manager

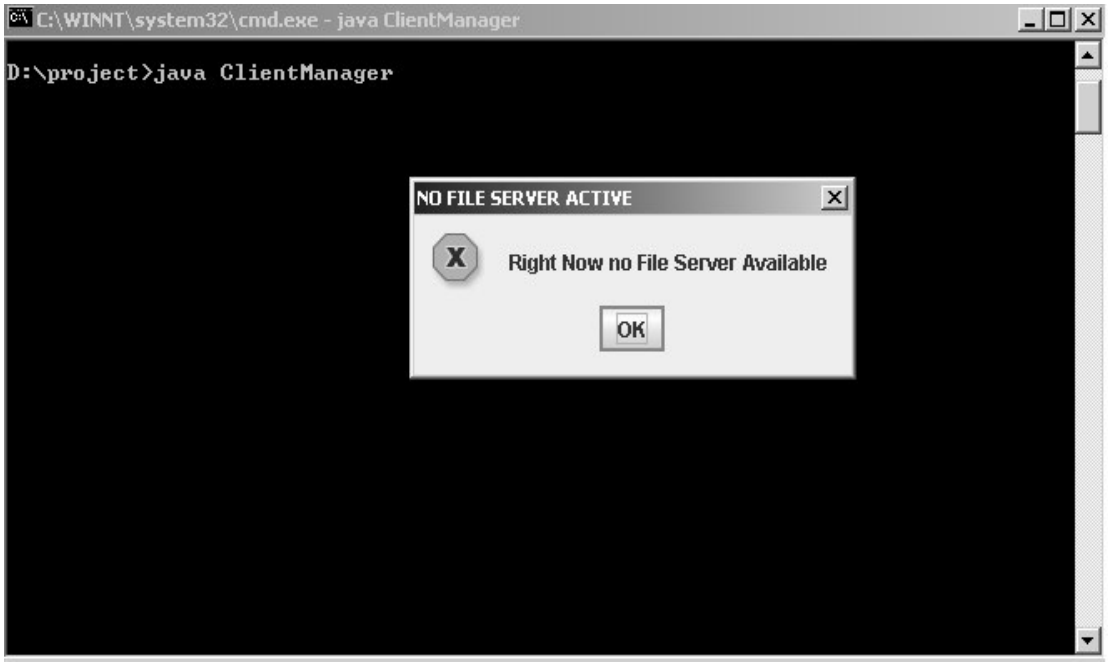


Figure 6.16 Acknowledgement Regarding the Unavailability of Active Home Servers

6.2.2 Observations

The two key parameters: File Mirroring and Message Passing of the prototype have been implemented in the current Fault Tolerant cluster configuration to make it possible for a client to avail the required service with some graceful degradation even in the situation when its domain Home Server becomes inactive as indicated in Table 6.1.

Table 6.1 Download Time for File Client.java

Type of Mode	Download time(milliseconds)
Normal	14781
Failure	20000

Client (192.168.3.22) issues request for the downloading of file Client.java to its domain active Home Server (192.168.3.22) and it become available to client in 14781 milliseconds. If HomeServer (192.168.3.22) becomes faulty then MainServerManager redirect this file access request to other active Home Server (192.168.3.16) and file becomes available to client in 20000 milliseconds.

The implementation detail of the prototype having the fault tolerance at cluster level has been discussed in this chapter. The next chapter includes the conclusion and the future scope of the enhancement of fault tolerance in grid environment.

Chapter 7 Conclusion and Future Scope

This thesis work provided an insight into Grid computing, Self-Management through Autonomic Computing, the fault tolerance approaches currently used in Cluster level of Grid and the problems found in these approaches. Detailed design of the prototype of the Cluster as a computing node of Grid its implementation has been given to depict the way with which the proactive fault tolerance can be incorporated into the cluster level of the grid.

7.1 Conclusions

Through this thesis, we have described multiple aspects of Grid Computing and introduced numerous concepts which illustrate its broad capabilities. Grid Computing is definitely a promising tendency to solve high demanding applications and all kinds of problems. Objective of the grid environment is to achieve high performance computing by optimal usage of geographically distributed and heterogeneous resources and to provide self-management capabilities to reduce the time and cost factor making the grid more reliable.

Grid self-management remains a challenge in dynamic grid environment. The Dynamic nature and complexity of Grid make it vulnerable to faults. Faults can occur in the Grid at any moment. At a conceptually finer level, clusters are often used as computing nodes in global Grid environments. The problems of management of Faults in grid are even more critical, because single clusters should (ideally) function autonomously without human intervention.

The Existing fault-tolerant approaches are inefficient because they are reactive and incomplete. They are reactive because they only deal with faults when they take place; they are incomplete because they only deal with certain types of faults. So a proactive approach is required to handle the faults i.e. to prevent faults to occur or to recover from the faults in a graceful way.

By using the agent based architecture the goal of proactive fault tolerance can be achieved. The agent oriented approach as specified in Autonomic computing framework deals with individual faults to provide Fault tolerance proactively. The prototype implemented in the thesis work shows the way in which a proactive fault tolerance approach can be incorporated at the cluster level of the Grid.

7.2 Future Scope of work

Monitoring is the heart of the architecture of fault tolerant system. The Heartbeat monitoring system as specified in the OGSA specification uses the 'heartbeat' to detect and report processes that have failed. The following are the shortcomings in the monitoring mechanism:

- It doesn't indicate the degree of urgency involved with the failure.
- HBM only informs if a process is alive or dead- not the process's actual health or state of being.
- It has no ability to resolve conflicts in perceived levels to enable the extended health check functionality.
- It doesn't take into account the environment and the past output.

The cognition system has been discussed in the thesis work which takes into account the past observations, the current values and the current environment for the monitoring. The characteristics of cognition system, affect and emotion thus would result in a more proactive fault-tolerant system.

References

[1] C. Kesselman., I. Foster, “Computational grids. In The Grid: Blueprint for a New Computing Infrastructure”, Morgan-Kaufman edition, 1999.

[2] I. Foster, C. Kesselman, eds., “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann, 1999.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", 2001.

[4] IBM’s Autonomic Computing- White Paper, “An architectural blueprint for autonomic computing”, June-2006, 4th addition.

[5] C. Kesselman., I. Foster. Globus “A metacomputing infrastructure toolkit”, journal Supercomputer Applications, 1997.

[6] Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., “Introduction to Grid Computing with Globus”, Redbook IBM Corporation, http://www.redbooks.-ibm.com/redbooks/pdfs/sg246895.pdf

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

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

[9] Duane Nickull “Service Oriented Architecture Whitepaper” Adobe Systems Incorporated, 2004.

[10] Jean-Christophe Durand, “Grid Computing a Conceptual and Practical Study” , November 8, 2004

[11] Gregor von laszewaski, Ian Foster, Argonne National Laboratory, “Designing Grid Based Problem solving Environments”, www.unix.mcs.anl.gov/laszewsk/-papers/cog-pse-final.pdf

[12] IBM Grid Computing, http://www-1.ibm.com/grid/grid_literature.shtml

[13] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (June 2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.globus.org/research/papers/ogsa.pdf.

[14] Rajesh Raman, Miron Livny, Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", University of Wisconsin , Madison, WI .

[15] David De Roure and others, "The Semantic Grid: Past, Present and Future", Proceedings of the 2nd Annual European Semantic Web Conference (ESWC2005), Volume 93, Issue 3, 2005.

[16] Dazhang Gu, Lin Yang, Lonnie R. Welch, "A Predictive, Decentralized Load Balancing Approach", ,Center for Intelligent, Distributed and Dependable Systems ,School of Electrical Engineering & Computer Science ,Ohio University, Athens.

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

[18] Ferreira, L., Bieberstein, N., Berstis, V., Armstrong, J., "Introduction to Grid Computing with Globus," Redbook, IBM Corp., http://www.redbooks.ibm.com-/redbooks/pdfs/sg246895.pdf

[19] M. Surridge "A rough Guide to Grid Security", Issue 1.1a, 2002

[20] Xianan Zhang, Dmitrii Zagorodnov, Matti Hiltunen, Keith Marzullo, Richard D**.** Schlichting, "An Agent Oriented Proactive Fault-tolerant Framework for Grid Computing", Universiry of Califonia, San Diego and AT&T Research Laboratory,2003.

[21] Floyd Piedad, M.H., "High Availability: Design, Techniques and Processes". Enterprise computing series: Printice Hall PTR,2002

[23] P. Horn, "Autonomic computing: IBM perspective on the state of information technology", IBM T.J. Watson Labs, NY, 15th October 2001.

[24] Julie A. McCann," Evaluation issues in Autonomic Computing", Department Of Computing, Imperial College London

[25] Kephart J. O., Chess D.M., "The Vision of Autonomic Computing", Computer, IEEE, Volume 36, Issue 1, 2003.

[26] R Sterritt, DW Bustard, "Autonomic Computing-a Means of Achieving Dependability?", Proceedings of IEEE International Conference on the Engineering of Computer Based Systems 2003

[27] D. F. Bantz and D. Frank, "Challenges in autonomic personal computing, with some new results in automatic configuration management," in Proceedings IEEE-2003

[28] Garlan D., B. Schmerl., "Exploiting Architectural Design Knowledge to Support Self-Repairing Systems" Proceedings of the 14th international conference on Software engineering and knowledge engineering. July 2002

[29] Kuo-Ming C., James A., Norman P., "A Framework for Intelligent Agents within Effective Concurrent Design", July 2001.

[30] Cheng S-W., Garlan D., Schmerl B., Steenkiste P., Ningning Hu., "Software Architecture-based Adaptation for Grid Computing", IEEE-2002.

[31] M. Agarwal and M. Parashar., "Enabling autonomic compositions in grid environments", 2003.

[32] Dashofy E. M., van der Hoek A., Taylor R. N., "Towards Architecture-based Self-Healing Systems", 2002.

[33] Roy Sterritt, and David F. Bantz, "Personal Autonomic Computing Reflex Reactions and Self-Healing", IEEE-2003.

[34] Garlan D., Schmerl B., Chan J., "Using Gauges for Architecture-Based Monitoring and Adaptation", 2001.

[35] The Globus Heartbeat Monitor Specification v1.0. [Online], http://www.fp.globus.org/hbm/heartbeat_spec.html

[36] D. DeCoste, S. G. Finley, H. B. Hotz, G. E. Lanyi, A. P. Schlutsmeyer, R. L. Sherwood, M. K. Sue, J. Szijjarto, and E. J.Wyatt, "Beacon monitor operations experiment DS1 technology validation report" , 2000.

[37] R. Stemtt, D.W. Bustard, "Towards an Autonomic Computing Environment", 2003.

[38] E. J. Wyatt, M. Foster, A. Schlutsmeyer, R. Sherwood, and M. K. Sue, "An overview of the beacon monitor operations technology," 1997.

[39] R. Sterritt, "Pulse monitoring: Extending the health-check for the Autonomic Grid," IEEE-2003.

[40] R. Sterritt, D. Gunning, A. Meban, and P. Henning, "Exploring autonomic options in an unified fault management architecture through reflex reactions via pulse monitoring", IEEE-2004.

[41] D. A. Norman, A. Ortony, and D. M. Russell, "Affect and machine design: Lessons for the development of autonomous machines", IBM Systems Journal, 2003.

[42] Thomas Röblitz, and rest, "Autonomic Management of Large Clusters and Their Integration into the Grid", Journal of Grid Computing (2004), Springer 2005

[43] T. Roeblitz, F. Schintke and A. Reinefeld, “From Clusters to the Fabric: The Job Management Perspective”, IEEE-2003.

[44] F.D. Sacerdoti, M.J. Katz, M.L. Massie and D.E. Culler, “Wide Area Cluster Monitoring with Ganglia”, in Proceedings of the IEEE International Conference-2003.

[45] P. Uthayopas, J. Maneesilp and P. Ingongnam, “SCMS: An Integrated ClusterManagement Tool for Beowulf Cluster System”, 2000.\

[46] http://www.java.sun.com/

[47] www.globus.org

[48] www.grid-consortium.com

[49] www.ibm.com

[50] www.alchemi.net

1. Puneet Khurana, Inderveer Chana, "Towards an Autonomic Grid" published in National Conference, Pinnacle-2007, MIT College of Engineering, Pune, 13 March, 2007 **(Published)**.
2. Puneet Khurana, Inderveer Chana, "Making Grid Self-Healing", accepted in National Conference on Information Technology: Emerging Engineering Perspective and Practices, ITEEPP'07 2007, Thapar University 6-7 April 2007 **(Accepted)**.