Aglets Mobile Agent based Grid Monitoring System

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

Master of Engineering
in
Software Engineering

By:
Arindam Choudhury
(80731004)

Under the supervision
of
Dr. Inderveer Chana
Sr. Lecturer (SG)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

JULY 2009

Certificate

I hereby certify that the work which is being presented in the thesis report entitled, **"Aglets Mobile Agent based Grid Monitoring System"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. 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.

Arindam Choudhury

Roll No. 80731004.

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

Inderveer 03/07/09

Dr. Inderveer Chana
Sr. Lecturer (SG), CSED
Thapar University
Patiala-147004, Punjab.

Countersigned by:

03/07/2009

Head
Computer Science & Engineering Department,
Thapar University,
Patiala.

RK Sharma 7/7/09

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

Acknowledgement

I would like to express my sincere gratitude to my supervisor Dr. Inderveer Chana for her constant encouragement and valuable guidance throughout my research. She has instilled in me a passion towards scientific research that will prove to be vital as I continue to pursue my academic interests. She has prepared me well for thesis work and has worked hard to promote my work. I am thankful for her continual support, encouragement, and invaluable suggestions throughout the thesis work. She always offered words of encouragement when I was feeling down. She not only provided me help whenever needed, but also provided me the resources required to complete this thesis report on time.

I am also grateful to Mr. Luca Ferrari, admin of Aglets mobile agent project, for his valuable support and advice in installation, working and understanding the Aglets mobile agent platform.

I would also like to thanks users of gt-user mailing list specially Dr. Jim Basney and Dr. Charles Bacon for helping me in installation of Globus Toolkit-4.2.1.

I would like to say special thanks to Dr. Rajesh Kumar Bhatia, Head, Computer Science and Engineering Department for giving me an opportunity to do this work.

I am also thankful to Dr. Seema Bawa, Professor, Computer science and Engineering Department As well as Dr. Maninder Singh, Asst. Professor, Computer Science and Engineering Department for providing me required help for the completion of this work.

My heartfelt thanks go to my parents, to my brothers and to my friends, especially Darshan, Jarrar, Paritosh, Gaurav, Suman, Nandi, Tirthraj and Ms. Shashi (research scholar). This work could not have been accomplished without their support.

Arindam choudhury

Arindam Choudhury

Abstract

Grid offers an optimal solution to problems requiring large storage and/or processing power. Grid provides direct access to computers, data, software and many other resources. Sharing between these resources is highly controlled and done with consensus of both resource providers and consumers.

Grid is not centrally controlled and resources can leave or join the grid environment at any time. The resources used in grid can be physically dispersed, heterogeneous and can span different administrative domains. Due to these reasons the interaction between these resources can result in failures. Besides this, protocol incompatibilities, security incompatibilities, network faults further contribute to failures. Moreover fundamental resources of grid are computers which may suffer from hardware and software problems also. Therefore, a mechanism is required to monitor these resources to check their availability, information gathering and reporting their failures.

To address this problem, mobile agents can be used. Mobile agents reduce network use, overcome network latency and can execute autonomously and asynchronously. Mobile agents can also adapt dynamically with the changes in environment. They are naturally heterogeneous, robust and fault tolerant. Therefore, mobile agents are an obvious and favorable choice for creating grid related applications.

In this thesis, a mobile agent based monitoring system for grid environment has been proposed. The proposed monitoring system is based on Aglets mobile agent platform. Hyperic SIGAR has been used to generate resource information. MySQL has been used as backend. Frontend has been designed and developed using PHP. The proposed system has been successfully deployed and tested in Research Lab of Computer Science and Engineering Department, Thapar University.

Tables of Contents

List of Figures

Figures **Pages**

List of Tables

Chapter 1 Introduction

Grid computing provides optimum solution to problems requiring large processing power and storage by supporting computation across multiple administrative domains [1]. It offers a way of using the resources optimally inside an organization or between the organizations by supporting multiple administrative policies and security authentication and authorization mechanisms to enable sharing over network and Internet. This chapter introduces grid computing, discusses about its characteristics and its architecture. It also gives an overview of grid Resource Management and resource monitoring.

1.1 About Grid Computing

Performance of computing can be improved by using a better algorithm, using a faster computer or dividing the calculation among multiple computers [2]. Algorithms can be fine tuned to a limit. Faster computers are costly and there is also a limitation in speed of computer. The best way to improve the performance of computing is dividing the calculation among multiple computers. Desktop computers (like most others) are only utilized about 5% - the CPU stays idle most of the time. Grid computing uses this idle time and links many machines together to perform massive tasks that previously only super-computers could do. Traditional distributed computing can be characterized as a subset of grid computing. Distributed Computing normally manages or pools computer systems which are limited in their memory and processing power. On the other hand, grid computing is concerned with efficient utilization of a pool of heterogeneous systems with optimal workload management utilizing computational resources (servers, networks, storage, and information) acting together to create large pool of computing resources.

The computational grid can be compared to electric power grid. As power grid distributes electricity to dispersed remote places, grid computing enables distribution of computing resources. Grid computing has been researched in universities and institutions for long time. With creation of sophisticated policies and standards, grid computing is spreading and used in many scientific and engineering organizations. Engineers, biologist, earth scientists, high energy physicists, animation artist are using

grid computing for solving compute or data intensive problems. Figure 1.1 shows generic view of World-Wide Grid [3].

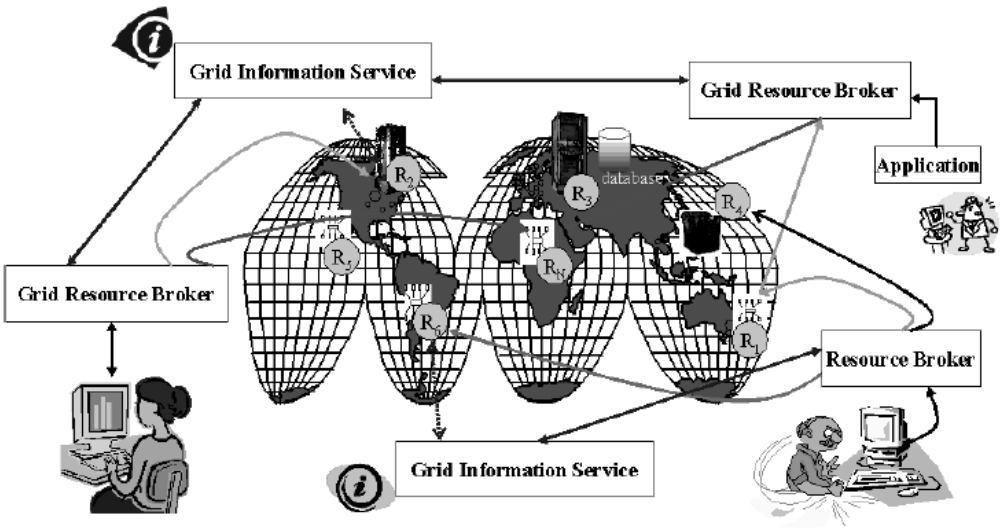

Figure 1.1 Generic View of World-Wide Grid (WWG) Computing Environment [4].

In grid computing processing power and storage capacity of physically dispersed computers are used to solve problems. Problems are divided into sub jobs and submitted to different computers connected though network or internet. After completion of these sub jobs, results are collected and combined to get the final result.

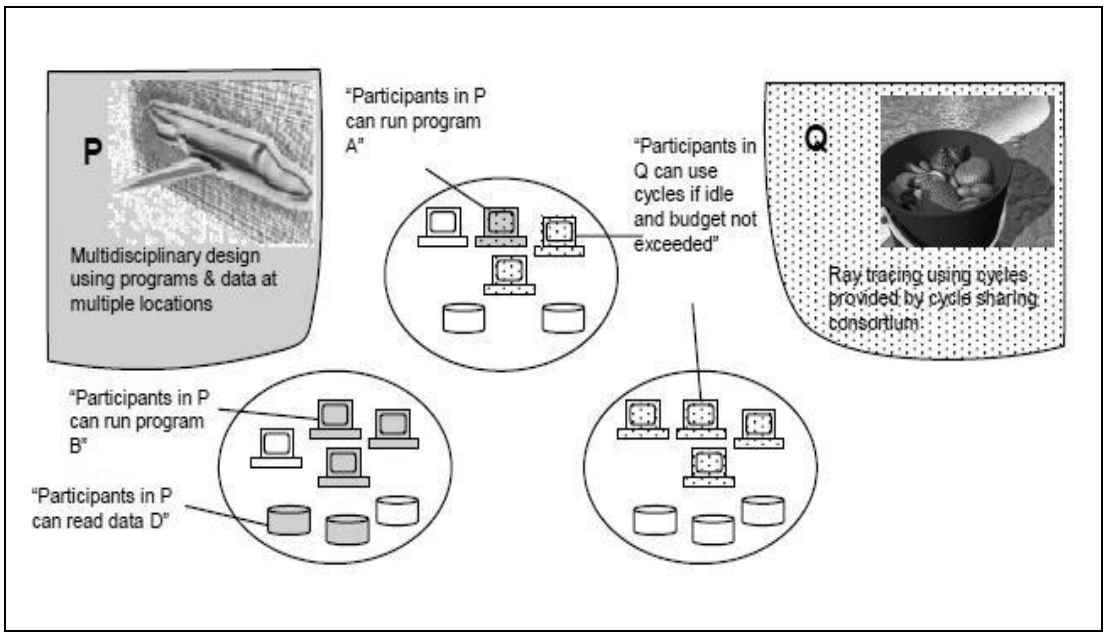

Figure 1.2 Three Actual Organizations are Taking Part in Two Virtual Organizations [5].

It provides direct access to computers, data, software and many other resources, which are heterogeneous in nature and lie in different administrative domains [5]. The

sharing policies of these resources are different. To accommodate these resources, the grid has to support a common mechanism for communication between these computers. The sharing must be highly controlled and done with the consensus of both resource consumers and resource providers. Virtual Organizations [5] are formed using these sharing rules. An organization can participate in two or more VOs. Figure 1.2 depicts actual organizations participating in two VO. It shows organizations taking part in different VOs to provide different services.

Grid computing should support capabilities to [6]:

- Remote storage, replication and publication of data sets.
- Access authorization and uniform authentication, uniform access to remote resources.
- Publication of services and cost.
- Discovery of suitable computer resources.
- Scheduling, submission, monitoring and migration of jobs.
- Movement of code/data between grid nodes.

Security is an important aspect of grid computing. Authentication, authorization, encryption and non-repudiation also used in grid computing. Besides these grid uses following security mechanisms [7]:

- Single sign-on: support authentication to a large number of grid resources on behalf of the user or resources. It creates temporary credentials (secure proxy) that provide authentication.
- Delegation: one grid entity act on behalf of another grid entity. Delegation must be handled carefully to resist the misuse.
- Community authorization: using this virtual organizations define policies to access resources.
- Secure execution: provide a service using which untrusted applications can run on trusted environment.

1.1.1 History of Grid

A decade ago, computers were expected to use local resources and data. But the scenario changed when system got interconnected through network. An expectation to use the ideal computing resources and data storage to fulfill a range of application

needs had been generated. The designing, building and deploying of distributed computing has been researched over many years. These researchers implemented middleware, libraries and tools that allow cooperative use of distributed resources. This approach of computing has been given different names, such as metacomputing, scalable computing, global computing and lately as grid computing [8] [9].

- The First Generation of Grid Computing: In early to mid 1990s grid efforts started to link the supercomputing sites. The FAFNER and I-WAY [10] are two well known projects that show the pathway to build grid. FAFNER was built to facilitate factoring via web of RSA factoring parallel algorithm. I-WAY helped to unify the resources of large US supercomputing centers. I-WAY used I-POP as gateway servers. I-POP software environment is used in the servers to overcome issues concerning heterogeneity, scalability, performance and security. I-POP servers are accessible via the Internet. The well known grid middleware Globus Toolkit is derived from the I-WAY project.
- The Second Generation of Grid Computing: The three main issues that must be addressed in grid are scalability, heterogeneity and adaptability. Grid middleware provide abstraction over resource heterogeneity and makes the grid scalable and adaptable. Usage of common standards also provides a solution to heterogeneity. Globus [11], a widely used grid middleware, enables the construction of computational grid. It supports wide range of application and programming paradigms. Globus is modular and constructed as a layered architecture. Globus provides GRAM to allocate, monitor and control computational resources. Globus uses GridFTP for data access. Globus further supports authentication and related security support, Lightweight Directory Access Protocol (LDAP), Global Access to Secondary Storage (GASS), Globus Advanced Reservation and Allocation (GARA).

  Legion [12] another popular grid middleware, provide infrastructure to grid computing. But it differs from Globus as it encapsulates all components as objects. The OMG (Object Management Group) [13] in 2001 introduced CORBA [14] in grid through “Software Service Grid Workshop”. Jini [15] & RMI is a CORBA based distributed computing environment.

  In grid, resources need to be discovered, monitored, and scheduled. The jobs submitted needed to be queued and batched. Condor, Portable Batch System

(PBS), Sun Grid Engine (SGE), Load Sharing Facility (LSF) are some of the batch and scheduling system created in this generation grid computing. Nimrod/G [16] is a grid enabled resource management and scheduling system. Nimrod/G supports user defined budget constraints and deadlines.

In the second generation of grid computing, the core technology for building a grid evolved. Besides that additional tools were also developed to support resource and job management.

- The Third Generation of Grid Computing**:** Third generation of grid computing aims to the reuse of existing resources and information systems, and to assemble these resources. The use of service-oriented model and metadata are two main characteristics of third generation grid computing. The concept of "Virtual Organization" and "Distributed Collaboration" were adopted. This generation of grid implements an autonomic system which can configure and reconfigure itself dynamically, implement open standards, recover from malfunction and optimize use of resources. It used service-oriented architecture like web-services, Open Grid Services Architecture (OGSA) [17], and agents. Web services support third generation grid requirements as it supports service-oriented approach and use standards to facilitate the information aspects. OGSA supports creation and maintenance of Virtual Organizations. Agent based computing provide autonomy to the dynamically changing grid environment.

The evolution of grid is still in progress. Next sub section discusses about different kind of grids.

1.1.2 Grid Types

Grid provides different type of services and resources [18]. Depending upon the design objectives and target applications grid computing can be broadly categorized in different types. The different types of grids are discussed below:

Grids can be categorized by the type of resources they share. The grids who shares different type of resources, they fall into more than one categories of grid given below:

- Computational Grids: In this type of grid primarily CPU resources are shared. Examples of computational grids are TeraGrid, SETI@home, and Sun Grid Compute Utility etc.

- Data Grids: Data resources such as result of experiments are shared between users. Data grids handle large amount of distributed data. Examples of data grids are QCDGrid and LHC computing grid.
- Storage Grids: Access to enormous amount of storage space is provided by storage grid. One of the biggest examples of storage grid is Amazon S3.
- Equipment Grids: Equipment grids share accesses to physical resources such as astronomical telescope are shared in eSTAR project.

Grid can be classified by the geographically distribution of their resources.

- Internet-scale Grid: In this type of grid anyone with access to internet can participate. SETI@home and World Community Grid are examples of Internet-scale grids.
- Virtual Organization-scale Grids: A VO-scale grid contains several academic or corporate entities. Most grids fall into this category e.g. TeraGrid, QCDGrid.
- Local Grids: Local grids are deployed within one organization. No other organization has access to this grid. For example, the render farms used to produce animated films such as Toy Story are a form of local grid.

Services provided by a grid are the most important thing. Data grid provided access to result of a large physics experiment. Some grid may not be able run some applications due to platform or licensing restrictions. Therefore, as grid becomes more prevalent, more initiatives such as Network.com which provides a catalogue of applications that can be run on the Sun grid computing facility can be seen.

At the moment, the most prevalent types of service offered by grids are forms of graphical rendering, scientific simulations and web applications.

1.2 Grid Architecture

Grid needs special architecture to cope up with heterogeneous resources and technologies. The architectures are discussed below:

1.2.1 Grid Protocol Architecture

The layered architecture of grid computing [5] is built on the TCP/IP protocol and services. The layers here are conceptual and do not imply constraints. Common mechanisms, interfaces, schemas, and protocols are defined in each layer, using which sharing relationship can be established, and computational power and data storage can

be shared over network. Figure 1.3 below shows the different layers of grid architecture and shows its relationship with TCP/IP protocol architecture:

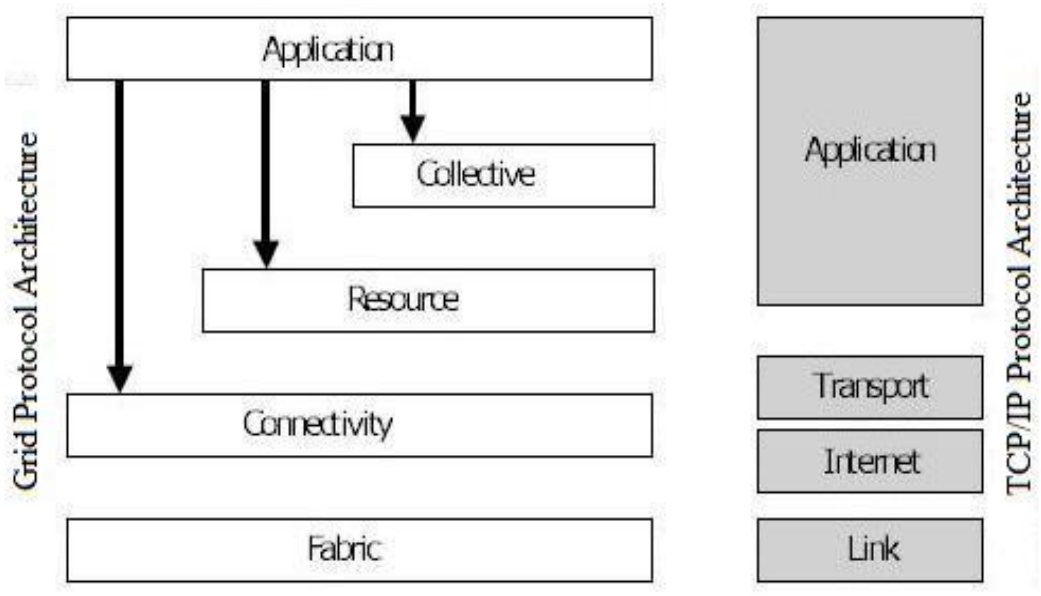

Figure1.3 The Layered Grid Architecture and its Relationship with TCP/IP Protocol Architecture [5].

The grid architecture consists of five layers. These layers are Fabric, Connectivity, Resource, Collective and Application. The brief discussions of these layers are given below:

- Fabric Layer: Fabric layer [5] defines set of operations to be performed on local resources to initiate sharing relationship at higher level. These operations are local and resource specific. The more sophisticated operations implemented in this layer, the more flexible relationship can be obtained in the higher level. Fabric layer should implement function to report structure, state and capabilities of the local resources, and resource management mechanisms.
- Connectivity Layer: Connectivity layer [5] defines the core communication and authentication protocols. Communication protocols are drawn from the Internet, transport and application layers of TCP/IP protocol stack. Authentication protocols provide security aspects of grid resource sharing. Authentication protocols also supports for communication protection.
- Resource Layer: The resource layer [5] defines protocol, which are built on connectivity layer protocol, for secure negotiation, control and monitoring of

individual resources. Protocol of this layer calls functions of fabric layer to access and control local resources. These protocols can be divided into information protocol, used to obtain information about state and structure, and management protocol which used to negotiate resource sharing specifying resource requirements and operation to be performed.

- Collective Layer: The collective layer [5] captures interaction between collections of resources where resource layer focused on single resource. This layer provide persistent services with associated protocols to support directory services to discover existence and properties of virtual organization and resources, co-allocation, scheduling, and brokering services, monitoring and diagnostics services, and many other services to facilitate data replication, community authorization, workload management etc.
- Application Layer: Application layer [5] is the final layer of grid architecture and it comprises of the user applications that operate within a virtual organization.

Grid protocol architecture defines the protocol should be used in the various layers. But an architecture is needed which can deal with the various services provided by the grid. OGSA is such an architecture which defines the creation of services and how the services can be manipulated.

1.2.2 OGSA: Open Grid Services Architecture

Open Grid Services Architecture (OGSA) [19] is a proposed set of standards for ensuring quality of service across a grid computing network. It designed for service oriented grid computing by Global Grid Forum (GGF) [20]. The objectives of OGSA are to manage resources across distributed heterogeneous platforms, delivery of seamless Quality of Service, and providing a common base for autonomic management solutions. The OGSA [17] defines open and published interfaces. To facilitate interoperability grids must use standard interfaces and protocols. OGSA extends web services for grid. It exploits industry standard integration technologies.

OGSA extends web services for grid computing. OGSA manages resources in grid [13] by providing:

- Scalability: OGSA provides scalability through hierarchical management of grid resources.

- Interoperability: OGSA use standard interfaces between heterogeneous resources. General IT management standards are also standardized.
- Security: security is deployed by using authentication, authorization, access control, access policy. Management also ensures it's own integrity by using security mechanism on management tasks.
- Reliability: reliability is achieved by not forcing a single point of failure. Managers are allowed to manage multiple resources and manageable resources are allowed to manage by multiple manager.

The OGSA architecture is shown below in the Figure1.4. The OGSA architecture comprises of four main layers. Short descriptions of these layers are given below:

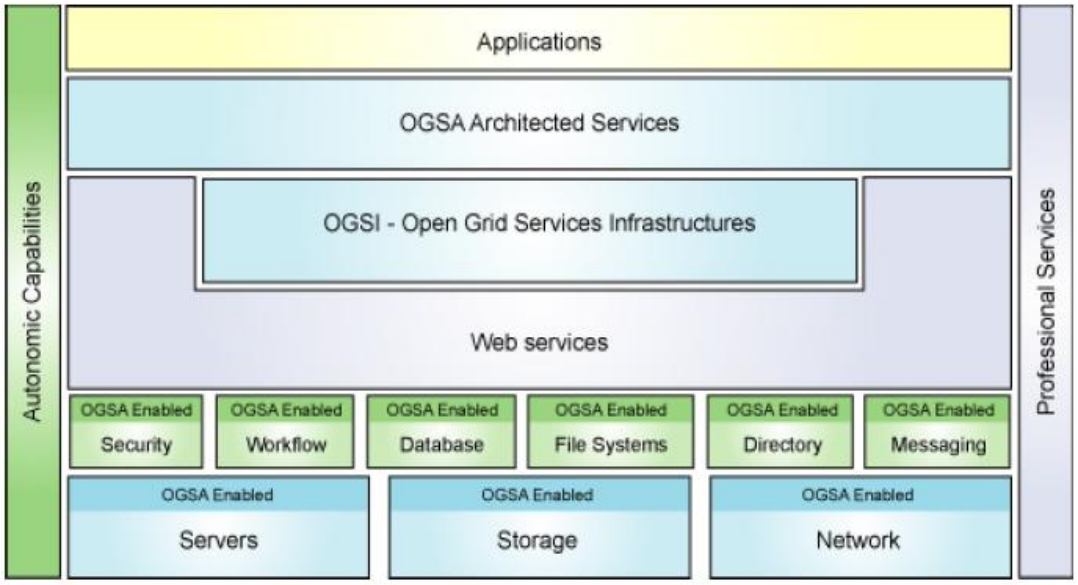

Figure1.4 The OGSA architecture [19]

- Resources- Physical resources and logical resources: In grid, resources are not limited to just processor. Resources like processor, servers, storage and networks are physical resources. Logical resources are above the physical layer and provide additional function by virtualizing and aggregating the physical resources. General purpose middleware provide these abstract services on top of physical grid.
- Web Services layer: In the second layer of OGSA both physical and logical resources are modeled as services using Open grid Services Infrastructure (OGSI). OGSI built on standard Web services technology and exploits mechanisms of Web services like WSDL and XML to specify standard interfaces, behaviors and interactions for grid resources.

- OGSA architected grid services layer: In this layer many grid services, architected with OGSI extensions, are defined in the area of program execution, data services, and core services.
- Grid Application Layer: In this last layer of OGSA, the applications that use OGSA architected grid services of level three are defined.

Grid shares different kind of resources and exploits them. Grid is not static; any grid node or resource can join or leave at any time. So, a mechanism is needed to manage these nodes or resources to enable proper sharing. Grid Resource Management is designed to fulfill this goal.

1.3 Grid Resource Management

Resource management in traditional computing systems is a well-studied problem. Grid Resource Management (GRM) is distinguished by the fact that the managed resources span administrative domains. This distribution can present problems due to heterogeneity in the way that similar resources are configured and administered. Different organizations operate their resources under different policies; the goals of the resource user and the resource provider may be inconsistent, or even in conflict. Further complicating the situation is the fact that grid applications often require the concurrent allocation of multiple resources.

The core goal of the resource management [21] [22] is to establish a mutual agreement between a resource provider and a resource consumer by which the provider agrees to supply a capability that can be used to perform some task on behalf of the consumer.

Resource management is the process of managing available resources and system workloads accordingly [23] to uniformly utilize resources of a grid. Grid resource management system basically address to principle needs given below:

- Available system resources should be maximally (or optimally) used in order to balance the distribution of the tasks over the different computing nodes.
- When events such as local congestion, a node failure or communication failure occurs, it is necessary to manage (re-)allocation of jobs and resources accordingly.

Resource management involves security and fault tolerance along with scheduling. GRM manages how the resources are allocated, assigned, authenticated, authorized,

assured, accounted, and audited. Resources can be traditional resources like compute cycles, network bandwidth, memory space, a storage system and also services like data transfer, simulation etc.

1.3.1 Classification of RMS

Classification of RMS is based upon grid type, resource namespace, resource information (discovery, dissemination), scheduling model and scheduling policy [22] [23].

- Grid Type: Grid resource management system deals with the management and distribution of CPU power over network in the case of Computational grids. Where in the case of Data grids GRM looks after proper distribution of experimentation results. So with the grid type the resources and objective of a grid resource management system changes accordingly. The aim and work policy of GRM also changes when it switches to Local grids from Internet-scale grids.
- Grid Namespace: GRM provide naming to resources and it helps in proper management of the resource. This naming affects the resource discovery, resource dissemination and it also affects the structure of database storing resource information. Naming process of a resource can be done using Relational, Hierarchical or Graph based namespace technique.
- Resource Information: Gathering resource information is an important aspect of grid resource monitoring system. Without this information, users of grid cannot interact with the resources properly as they would be blind about the state and structure of resources. GRM uses two procedures to gather system information. These procedures are resource discovery and resource dissemination.
- Scheduler Organization: One of the most important parts of GRM is schedulers. Schedulers take decisions on using and managing resources. Schedulers are deployed to facilitate uniform and balanced use of resources. Schedulers can be classified into three types, centralized, hierarchical and decentralized, depending upon the organization scheduling controllers.
- Rescheduling: Resources of a grid need to be rescheduled as a result of current schedule re-examination and re-ordering of job execution. Jobs are re-ordered to maximize resource utilization, job throughput etc. Rescheduling can be done in two ways: periodic/batch rescheduling and event-driven/online rescheduling.

- Scheduling Policy: Scheduling policies are used to determine relative ordering of resources and jobs when rescheduling. Scheduling policies can be divided in two categories depending how they can be alter by the entities outside the RMS: fixed and extensible.

Though grid has GRM deployed to manage resources, sharing and interaction between heterogeneous resources may lead to errors and failures in grid. Thus a monitoring system is essential.

1.3.2 Monitoring in Grid Environment

To achieve high-performance grid, it is critical to monitor and manage the system. Monitoring data determine the source of performance problems and help to tune the system for better performance. This data is also used by fault detection and recovery system to determine if a server is down, these data also help to make decision like restarting the server or redirecting service requests elsewhere. Prediction model use these data to predict performance, schedulers use these predictions to determine which resource to use. As grid systems are big, complex and widely distributed, it becomes more important to deploy automated monitoring and management systems.

1.4 Organization of the Thesis

Chapter 1 gives an overview of grid computing. It discusses various aspects of grid computing and gives historical pathway through which grid computing has evolved. It also discusses about the architecture of grid computing: the protocol architecture and the OGSA architecture. Lastly it describes Grid Resource Management system and resource monitoring. The rest of the thesis is organized as follows:

Chapter 2 discusses about faults in grids and methodology of mitigation if failure occurs. It provides architecture and taxonomy of monitoring system and compares some monitoring systems available. This chapter also introduces mobile software agents and discusses its advantages.

Chapter 3 formulates the research problem. It discusses about the objective of this thesis and methodology to achieve those.

Chapter 4 details the design and implementation of the proposed monitoring system. It discusses topology and architecture of the proposed monitoring system. It describes all the technologies used to implement the proposed monitoring system.

Chapter 5 discusses the deployment details and also provides the test cases and test results of the proposed monitoring system.

Chapter 6 concludes the thesis work and discusses about future work.

Chapter 2 Literature Review

This chapter discusses about failures in grid and procedures to mitigate these failures. Then it discusses about grid monitoring systems, its architecture, and various available monitoring systems. It also provides a comparative study on these monitoring systems. Finally it introduces mobile agent paradigm and discusses about its advantages over traditional computing.

2.1 Grid Monitoring

Monitoring is crucial in a variety of cases such as scheduling, data replication, accounting, performance analysis and optimization of distributed systems or individual applications, self-tuning applications and many more. Also, given the increasing number of grid resources, real-time monitoring of their availability and utilization is becoming essential for effective management, particularly regarding the detection of faults and bottlenecks and in some cases even their automatic resolution. Finally, identifying patterns of utilization may form valuable input for long-term resource planning of grid infrastructures.

Grid monitoring differs significantly from traditional monitoring of computer-related resources [24]. Grid monitoring is characterized by requirements including, among others, scalable support for both pull and push data delivery models applied over vast amounts of current and past monitoring data that may be distributed across organizations. In addition, a monitoring system's data format has to balance between extensibility and self-description on one hand and compactness on the other. The former is required to accommodate the ever expanding types of monitored resources, whereas the latter is a prerequisite for non-intrusive and scalable behavior. The problem is further complicated by the continuous evolution of grid middleware and the lack of consensus regarding data representation, protocols and semantics, leading to ad hoc solutions of limited interoperability. Existing proprietary network and host monitoring applications lack the openness required for interoperability and customization, while they also impose significant financial costs. Few of the equivalent open source projects have a potential and in fact some of them are actually employed for grid monitoring.

2.1.1 Faults in Grid

One of the biggest problems in grid computing is failures. The main reasons of grid failures are [26]:

- As grid uses huge number of hardware and software components, the probability of occurring failures increases with the size of the system.
- Grid is extremely dynamic; components join or leave the system all the time.
- Grid components are extremely heterogeneous, interaction between these components result in failures.
- Grid is not centrally controlled. Grid nodes are unstable and can be shutdown or damaged anytime.
- Grid resources span different administrative domain, the sharing and communication policies are different at different administrative domain.
- To combine different type of resources and applications, grid has to support various kinds of protocols and interfaces, the interaction between these protocol and interfaces may also lead to failures.

The fundamental building blocks of grid are mainly computers. Computers show different kinds of hardware and software failures [26]. As the availability of computers are not hundred percent all the times, when the computers are joined to create grid, the overall availability decreases proportionally with the number of computer attached to it. As the resources of grids are joined together using network, so failures in network are also a cause of grid faults. Thus failures in grid can be classified into these categories:

- Hardware Faults: These types of faults occur due to the manufacturing fault of the hardware. This can be result of a faulty CPU, faulty memory or faulty storage devices etc.
- Software Faults: These faults are result of human errors. No software is error free. This may be due to bugs, memory leak, and numerical exception. Configuration errors in software may also cause this type of error.
- Network Faults: Recent advances in networking devices have decreased the network faults, but they are still prevalent. Faults in network cable, routers, switches and heavy traffic in network are some reasons for network faults.

Besides these, grid encounters two more types of failures [27]:

- Interaction Faults: Protocol incompatibilities, security incompatibilities, policy problems, timing overhead results in this type of problem.
- Omission Faults: These faults happen due to the dynamic nature of grid nodes as some resources may become unavailable.

Fault tolerant computing had been researched through past few decades. Hardware fault tolerance can be achieved by creating a system which have MTBF measured in decades or centuries. This is done using [26]:

- System is decomposed hierarchically into modules.
- Modules are designed to have Mean Time before Failure (MTBF) in excess of a year.
- Each module is made fail-fast that it works right or stop working.
- Keep track of modules using heartbeat or watchdog timer. This helps to detect any failure promptly.
- Extra modules are configured to pick up the load of failed modules. The time between failure and pick up should be very small.

Software fault-tolerance [28] is based on the traditional hardware fault-tolerance approaches. Software fault-tolerance can be achieved by:

- Recovery blocks which are created using various implementations of the same algorithm. Any component of the system is composed with 'n' number of such recovery blocks acting as alternatives. An adjudicator manages these blocks. If the primary block fails, the adjudicator rolled back the state of system and tries the alternative secondary block.
- N-version software create modules which are made with up to N different implementations that accomplish the same task in different ways. Each version executes the problem and the result is selected using voting.
- Self-Checking software technique is used in highly reliable and safety critical systems. It includes rollback recovery methods. It searches heap finding and corrects data defects.

In a survey [29] about grid failures, it has been found that 76% of the responses said that the main kind of errors are due to environmental configuration errors, 48% of the responses point out middleware as the main source of errors, 43% of the responses

said the main source of error is application errors and 34% of the responses said that it is hardware errors. Figure 2.1 depicts the result of this survey.

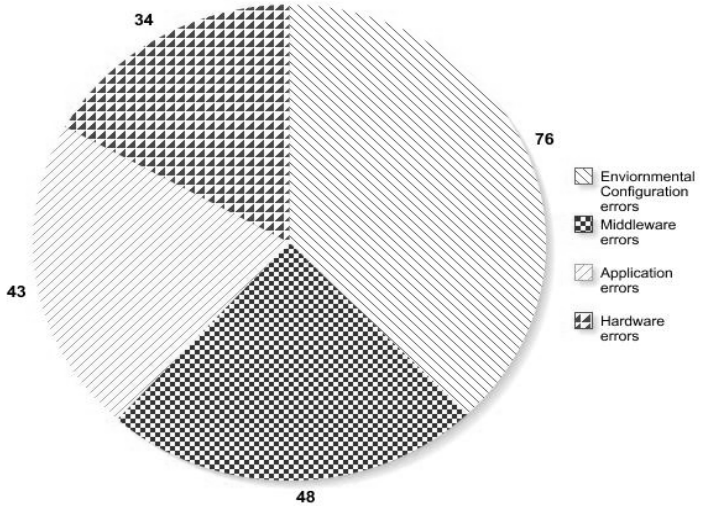

Figure 2.1 Pie Chart Depicting Major Sources of Failures in Grid

Grid developers provide abstraction in the middleware that hide the details of software and hardware components. This works great when everything goes right as users do not need to interfere with the details. But when the system fails, users need to dig down the abstraction to find out the faults. Identifying the root cause of the failure is also very difficult; as user diagnosing the failure may either not has the right authority or may not have any prior knowledge about working of the component.

Faults are inevitable in grid but they can be mitigated. Most hardware and software failures reported are transient [26]. They can be overcome by retrying the process in the same machine with memory correction and program state reinitialized; the processes usually do not fail for the second time. There are hard failures also where the jobs need to be shifted to another machine. To achieve fault-tolerance in grid computing the following techniques have been used traditionally:

- Retrying: This is the simplest form of fault tolerance. In this the job is restarted from scratch after failure.
- Replication: In this fault-tolerant scheme the same job is submitted to different nodes. After completion of job the result is chosen by polling the results which came from different nodes. If the number of nodes where the job is replicated is n then the result with n/2 or more votes is selected.
- Checkpointing: Checkpointing is used to store the state of process and the messages sent and received by the process during fail free execution time. When

failure occurs, the backup process restarts from the last saved state using checkpointing data.

So monitoring system is needed which can identify any occurrence of failure in grid environment and it provides needed mechanism to overcome it. It also provides an information system which supports user need by finding and keeping track of grid resources.

2.1.2 Concepts and Terminology

It is important to note that people use different terms to refer to, more or less, the same concepts based upon the type of monitoring. The terms are [25]:

- Entity: An entity is any useful networked resource. An entity is unique. It has a considerable lifetime. Typical entities are processors, memories, storage mediums, network links, applications and processes.
- Event: An event is a collection of data associated with an entity and represented in a specific structure. These data are timestamped and typed.
- Event Type: An event type is an identifier which uniquely maps to an event structure.
- Event Schema: An event schema or simply schema defines the typed structure and semantics of all events so that given an event type one can find the structure and interpret the semantics of the corresponding event.
- Sensor: A sensor is a process monitoring an entity and generating events. Sensors are distinguished in passive (i.e., use readily available measurements, typically from operating system facilities) and active (i.e., estimate measurements using custom benchmarks). The former typically provide OS specific measurements while the latter are more intrusive.

2.1.3 The Monitoring Process

Monitoring distributed systems, and hence grids, typically includes four stages [30] [31]:

a) Generation of events, that is, sensors enquiring entities and encoding the measurements according to a given schema.
b) Processing of generated events is application-specific and may take place during any stage of the monitoring process.

c) Distribution refers to the transmission of the events from the source to any interested parties.

d) Finally, Presentation involves further processing of received events. The overwhelming numbers of received events are processed to provide abstractions so that users can draw conclusions about the operation of the monitored system. A presentation is provided by GUI application or visualization techniques or using a real-time stream.

2.1.4 Requirements for Grid Monitoring Systems

Grid monitoring should implement some important general requirements. These requirements are discussed in this section. These requirements may vary considerably depending on use cases that are supported by a specific system [24].

- Scalability: Monitoring systems have to cope efficiently with a growing number of resources, events and users. This scalability can be achieved by introducing good performance and low intrusiveness. Good performance guarantees needed throughput in a variety of load scenario within an acceptable response time. The low intrusiveness is the key to widespread deployment of the system, scalability issues and application monitoring.
- Extensibility: A monitoring system must support various kinds of resources by supporting events generated by those resources. Desirable features of a monitoring system includes:
  i. an extensible and self-describing event encoding method.
  ii. an event schema service which allows controlled and dynamic extensions/modifications;
  iii. producer-consumer protocol that can accommodate new event types.

  At the same time, (i) and (iii) must be compact to minimize the previously described intrusiveness, which is so important for scalability.
- Portability: A monitoring system should be portable. The portability can be achieved by making the sensors supporting various resources on various platforms. The encapsulated measurements of generated events must be platform independent.

- Security: Monitoring should support security services like access control, authentication on certain scenarios. The transportation of monitoring information should be secure.

In addition, it is already mentioned that an event is a collection of timestamped typed data; hence a timestamp is required to allow consumers to estimate an event's freshness. So the time between the component of monitoring system and users need to be synchronized. In grid, the system participating are already need to be time synchronized, but monitoring system needs a much higher accuracy of clock synchronization. The time can be synchronized using Network Time Protocol.

2.1.5 Grid Monitoring Architecture

This subsection provides a brief overview of Grid Monitoring Architecture (GMA) [32] developed by the Global Grid Forum to encourage discussion and implementations of monitoring systems. GMA is not a standard. The main components of GMA are as follows as shown in Figure 2.2:

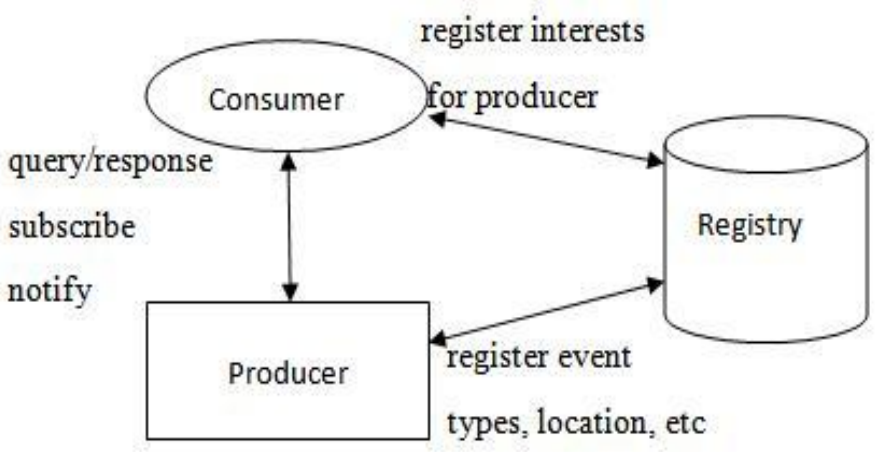

Figure 2.2 Grid Monitoring Architecture [24]

- A producer is a process implementing at least one producer Application Programming Interface (API) for providing events.
- A consumer is any process that receives events by using an implementation of at least one consumer API.
- A registry is a lookup service. Registry allows producers to publish the generated event types, and consumers to find out the events they are interested in. Additionally, a registry holds the details required for establishing communication with registered parties (e.g., address, supported protocol bindings, security

requirements). Monitoring systems those do not have events, registries can be useful to producer and consumer for discovering each other.

Interactions: As discussed earlier producers and consumers discover each other using the registry. After discovering they communicate with each other directly, not through the registry. GMA defines three types of interactions between producers and consumers [24]. Publish/subscribe is a three-phase interaction. Firstly subscription is made for specific event type. Then events are streamed from a producer to a consumer. At last the subscription is terminated. The establishment and the termination of subscription can be initiated by a consumer or a producer. A query/response is a one-off interaction. It is initiated by a consumer. A single producer responds to the consumer. The response contains one or more events. Finally, a producer can send notification to a consumer without any further interactions.

The GMA [32] defines a republisher and a schema repository in addition to the three core components.

- A republisher is any single component implementing both producer and consumer interfaces, as shown in Figure 2.3, for reasons such as filtering, aggregating, summarizing, broadcasting, and caching.

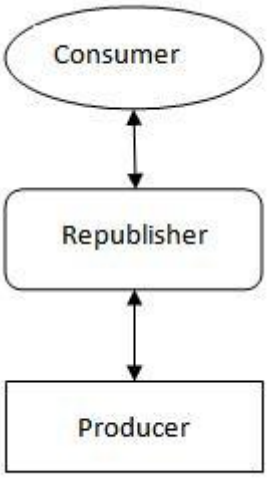

Figure 2.3 A Republisher [24]

- A schema repository holds the event schema. Schema repository is needed by the monitoring systems needs to support an extensible event schema. Schema repository implements interface for dynamic and controlled addition, modification and removal of any custom event types.

Republishers and the schema repository are optional components. But they are essential parts of any sophisticated monitoring framework. The schema repository may be part of the registry, but in any case these two components must be replicated and distributed to allow for distribution of load and robustness.

The GMA does not define implementation details. It does not define employed data model, event schema, protocol bindings, registry engine or other implementation issues. Probably the most important feature of the GMA is the separation of the discovery and retrieval operations. Consumer and producer discover each other from the registry and events are retrieved from producers or republishers.

2.1.6 Taxonomy of Monitoring Approaches and Frameworks

This subsection discusses taxonomy of monitoring systems. The taxonomy [24] primarily concerns on how the monitoring system implements GMA components. As shown in Figure 2.4, the categories of the taxonomy are named from zero to three. They are categorized on the basis of implementation of producers and republishers.

- Level 0: Monitoring systems of this category implements sensors and consumers. This is the simplest monitoring system. It does not have any producers. The flow of events is from sensors to consumers. The flow can be done in an on-line or an offline fashion. In the case of online event flow, sensors store the measurements locally and these measurements are accessed via application specific ways. The lack of producer APIs in the level zero enable distribution of events to remotely located components.

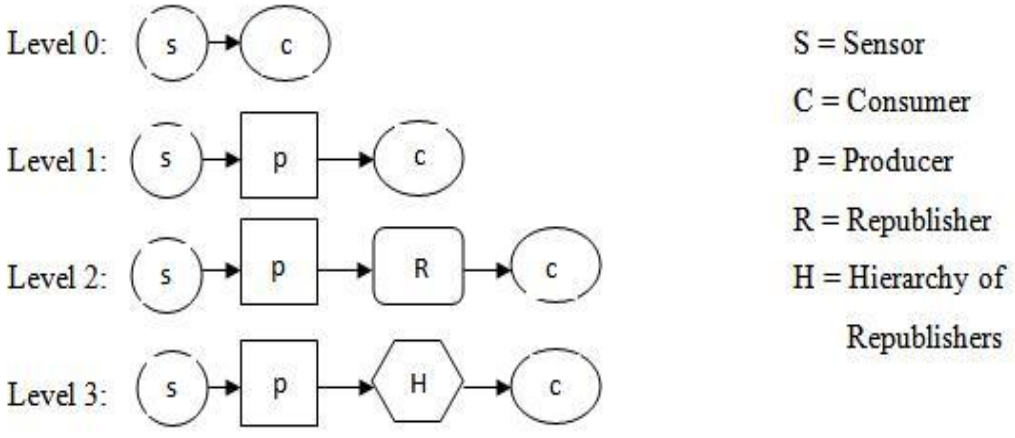

Figure 2.4 The Categories of Taxonomy of Monitoring Systems [24]

- Level 1: In first level systems, sensors can be implemented in two ways. They can be hosted with producer in the same machine or they can be implemented

separately. The events generated by the sensors can be accessed via generic API provided by producers in both cases.

- Level 2: The level two monitoring systems implements at least one type of republisher. The republisher used in this level has fixed functionality. Republishers providing different functionality are stacked upon each other. The stacking is done in predefined ways.
- Level 3: Level three monitoring system is highly flexible. It provides configurable republishers. The republishers can be organized in arbitrary structured hierarchy. Every node collects and processes events generated by lower level producers or republishers. This helps in producing customized views or preparation of higher-level events. The third level systems support scalability and can form standalone Grid Information Service.

The taxonomy includes a multiplicity qualifier to capture whether republishers in second level systems within an organization are centralized (i.e. one republisher), merely distributed, or distributed with support for replication. The types of entities are also referred by another qualifier. Whether a monitoring system can use another monitoring system's producers or republishers is denoted by the stackable qualifier.

Based on the above categories and qualifiers, considered systems are characterized using the form L{0-3}. {H,N,A,V,G}.[S], where the number denotes the level, the following letter denotes the nature of monitored entities (Hosts, Networks, Applications, Availability, Generic), and an optional S implies whether the system is stackable.

a) Level 0: Self-contained Systems

A "self-contained" monitoring system does not expose its functionality through a producer interface. Such a system can be used only in predefined and rigid ways (e.g., through a GUI). The considered self-contained system GridICE supports grid administrators in monitoring the availability (in the former) and utilization (in the latter) of grid resources, through web front-ends.

GridICE

GridICE (L0.G.S) [33] is developed as part of DataTag project. GridICE is also known as InterGrid Monitor Map and EDT-Monitor. It provides site and resource level information about status and utilization at Virtual Organization. It also provides

basic statistics derived from historical traces and real-time alerts. All information is provided through a web front-end.

GridICE has a centralized architecture. A main server extracts status information of grid and network services from a set of nodes periodically. Database management system is used to store information collected. PHP-based web front-end is used to provide information to the users. Nagios is used to monitor any newly discovered resources. Negious is an open source host and network service monitor.

b) Level 1: Producer-only Systems

As an example of a first level system, this subsection presents Autopilot, a framework for implementing self adapting applications.

Autopilot

Autopilot (L1.A) [34] can adapt changing environments using runtime tuning ability. Application adaptability can be achieved by real-time performance measurements. Autopilot's functionality is implemented in separate components. The components are sensors, actuators, clients and distributed name servers.

Sensors and actuators are used for reading and writing application-level variables. These sensors and actuators are described by property lists like name, location, type of variable measured/controlled etc. Sensors and actuators register themselves to a registry. Consumers or clients search registry for sensors using property lists.

An Autopilot client finds out sensors through the registry. Then the client subscribes for receiving their events. A client makes decision using application specific logic. Client can instruct an actuator to perform actions.

c) Level 2: Producer and Republisher Systems

The level two monitoring system includes one or more special purpose republishers. Level two systems can be divided into three subcategories [24]. The subcategories are given below:

- Centralized republisher (L2a)
- Distributed republishers (L2b)
- Distributed republishers with replication (L2c)

Hawkeye

Hawkeye (L2a.G) [35] is a monitoring and management tool for clusters of computers. Hawkeye uses some technology from Condor. It is available as a standalone distribution for Linux and Solaris.

In this monitoring system every participating node deploys producer and the producer is called monitoring agent. These agents periodically check host's state. Agents communicate with central manager which is a republisher.

The central manager indexes the current state of nodes so that fast queries can be executed. It stores information into a round robin database periodically. Information from the database can be accessed by command line utility, web and GUI frontends.

Mercury

Mercury (L2b.G) [36] is a generic and extensible monitoring system. It has built as part of GridLab project. Mercury is a Grid-enhanced version of the GRM distributed monitor of the P-GRADE graphical parallel program development environment.

Mercury consists of local monitors (producer), a main monitor and a monitoring service (republisher). Sensors are deployed in Local monitors as loadable modules. These sensors are used to collect information about the local node. Local monitors send information to the main monitor.

The monitoring service takes care of the queries of external consumers. When monitoring service receives a request, it validates the request against site policy. If the request is valid then the request is sent to the main monitor. Main monitor in turn request local monitors. The main manager gets results from the local monitor and sends result back to monitoring service. At last monitoring service forwards the results to the consumer. Several instances of main monitor can be used to allow load balancing.

NetLogger

The Network Application Logger Toolkit (NetLogger) (L2a.A) [37] is used for performance analysis of complex systems. NetLogger facilitates identification of performances bottlenecks.

NetLogger is built of four components. The components are an API and its library, tools for collecting and manipulating logs, host and network sensors and a front-end.

Any disk/network I/O requests and any time-consuming computations invoke NetLogger's API. The generated events are stored in a local file, syslog daemon or in a remote host.

NWS

The Network Weather Service (L2c.N) [38] is a non-intrusive performance monitoring and forecasting distributed system.

NWS is primarily built intended to support scheduling and dynamic resource allocation. Sensors (producers) are used at hosts to estimate CPU load, memory utilization, network bandwidth and latency etc. Sensors provide accuracy in measurements by combine use of passive and active monitoring methods. The stateless sensors are used to improve robustness and minimize intrusiveness. Sensor control process is used to manage sensors. Events of the sensors are sent to a memory service. Load predictions are created by the forecasting process which consumes events from the memory service. Memory service and forecasting process act as republishers. All components subscribe to an LDAP-based registry.

d) Level 3: Hierarchy of Republishers

This subsection focuses on third level monitoring systems. Third level systems feature producers and general purpose republishers which can form an arbitrarily structured hierarchy.

Ganglia

A ganglion (L3.G) [39] is an open source hierarchical monitoring system. Primarily Ganglia is designed for computer clusters but also used in grid.

Membership at the cluster level is done using a soft-state broadcast protocol. The memberships need to be renewed periodically or it will expire. All nodes deploy ganglia monitoring daemon. This monitoring daemon is multithreaded. The generated events are encoded using External Data Representation. The encoded events are broadcasted. Monitoring daemon listen to broadcasts of other nodes. Cluster's state is locally maintained by listening to these broadcasts. Monitoring daemon also replies to queries of other nodes of local cluster. As result of above discussed actions cluster's status is replicated among all nodes. The nodes act as the producers. This also helps in distribution of load. Ganglia does not use any registry. The database serves as the archives. New event type and querying producers and republishers are done by simple command line utilities.

Ganglia uses structured hierarchy of republishers. Republishers periodically store events in a round robin database. Information from the database is provided to higher level republishers on demand.

Globus MDS

The Monitoring and Discovery Service (L3.G.S) comes with the Globus Toolkit. Formerly this monitoring system is known as the Metacomputing Directory Service.

It constitutes the information infrastructure of the Globus toolkit. Globus MDS [40] is based on two core protocols. The first one is Grid Information Protocol (GRIP) which allows query/response interactions and search operations. The second one is Grid Registration Protocol (GRRP) which maintains soft-state registration between MDS components.

The MDS framework consists of three components. The components are information providers (sensors), Grid Resource Information Services (GRIS—producers) and Grid Index Information Services (GIIS—republishers). Lightweight Directory Access Protocol is used as a data model and both producers and republishers are implemented as backend.

Information providers generate events and these events are collected by the producers. Republishers and consumers get the events from producer using GRIP. GRRP is used by producers to register themselves to one or more republishers. Republishers are organized in hierarchy. Consumers discover producers through republishers using GRIP. Consumers submit queries to producers or republishers.

RGMA

The Relational Grid Information Services Research Group of the Global Grid Forum has built RGMA (L3.G.S) [41] as part of the EU DataGrid project. RGMA framework combines grid monitoring and information services.

In RGMA the producers or consumers need to instantiate a remote object i.e. an agent and need to invoke methods from appropriate API. Core relations are included in global schema. New relation can be created and dropped in global schema by the producers. Republishers are defined as SQL queries. Registry holds the relations and views provided by the producers and the republishers. The registry is centralized and the global schema is included in it. RGMA can be used as standalone grid information service.

The next subsection compares these monitoring systems.

2.1.7 Comparison of Monitoring Systems

Two approaches are used to compare the monitoring systems with each other. One is based on the systems overview with respect to components mapping to GMA, illustrated in Table 2.1, 2.2, and the second one is based on systems overview with respect to implementation and miscellany features which is depicted in Table 2.3, 2.4.

Features/project	AutoPilot	Hawkeye	Mercury	NetLogger
Classification	L1.A	L2a.G	L2b.G	L2a.H
Producer	Instrumented Applications	Agent	LM	Instrumented Applications
Republisher	n/a	Manager	MM,MS	Activation producer
Registry	Name service	Manager	MM	Registry
Archive	-	Round robin DB	DBMS	DBMS

Table 2.1 Systems Overview with Respect to Components Mapping to GMA- Part 1 [24]

GMA mapping refers to how a monitoring system implements all the entities described in the Grid Monitoring Architecture. It shows the difference of implementation of producer, republisher, registry, and archive. Table 2.1 and 2.2 shows the comparisons.

Features/project	NWS	Ganglia	MDS2	RGMA
Classification	L2c.G	L3.G	L3.G.S	L3.G.S
Producer	Memory host	Monitoring daemon	GRIS	Producer
Republisher	Predictor	Meta daemon	GIIS	Republishers
Registry	Name service	-	GIIS	Registry
Archive	-	RRD tool	-	DBMS

Table 2.2 Systems Overview with Respect to Components Mapping to GMA – Part 2 [24]

Tables 2.3 and 2.4 refer to the implementation issues and miscellany features. Implementation issues are compared using the language used, supported APIs, supported tools and whether the monitoring system is available through Internet. Java based monitoring systems provide less performance than a monitoring system developed using C/C++ languages. So language used provides indication about the hosts performance and overhead.

In miscellany key features and security is considered. The key features row shows the diversity between the monitoring systems.

The security row shows whether the monitoring system implements security features. Security is provided by authentication and authorization via Grid Security Infrastructure.

Features/project	AutoPilot	Hawkeye	Mercury	NetLogger
IMPLEMENTATION				
Language	C++	C++	C, Java	C/Python
API	C++	C++, Java	C	C,C++,Perl, Python, Java
Tools	Visualization GUI	Web front-end	Visualization GUI	Visualization GUI
Availability	Y	Y	Y	Y
MISCELLANY				
Key features	App steering	Cluster stats/management	Extensibility and adaptivity	-
Security	GSI	GSI	GSI	-

Table 2.3 Systems Overview with Respect to Implementation and Miscellany- Part 1 [24]

Features/project	NWS	Ganglia	MDS2	RGMA
IMPLEMENTATION				
Language	C	C	C	Java Servlets
API	C	CLI util	C, Corba, Perl	C,C++,Perl, Python, Java
Tools	Web front-end	Web front-end	Web front-end	Visualization GUI
Availability	Y	Y	Y	Y
MISCELLANY				
Key features	Load predictions	Cluster stats	Custom views	Virtual DBMS
Security	-		GSI	GSI

Table 2.4 Systems Overview with Respect to Implementation and Miscellany – Part 2 [24]

Agents can help to obtain autonomy in grid computing. Several grid projects have been developed using agent technology such as JAMM [42], MAP [43] etc. MAP is developed using mobile agent. The next section describes mobile agent in brief.

2.2 Mobile Agent in Grid Computing

Mobile agents help easy maintenance, design and implementation of grid systems. Software agents can be defined as a program that assists and acts on behalf. Software agents [44] [45] are of two types: stationary agents and mobile agents. Stationary agents execute on the system where it begins its execution, if it needs information from any other system then it communicates with an agent residing on that system.

Mobile agents are free to travel to any host in the network. Mobile agents should have the following mandatory properties [44]:

- Reactive – can sense changes in the environment and act accordingly.
- Autonomous – has control over its own actions.
- Goal-driven – is pro-active.
- Temporally continuous – is continuously executing.

and may possess any of the following orthogonal properties [44]:

- Communicative - able to communicate with other agents;
- Mobile - can travel from one host to another;
- Learning - adapts in accordance with previous experience;
- Believable - appears believable to the end-user.

Mobile agent provides a powerful, uniform paradigm for network computing that can revolutionized the design and development of grid computing. The next subsection describes the mobile agent paradigm.

2.2.1 Mobile Agent Paradigm

The figure below shows pictorial view of mobile agent paradigm:

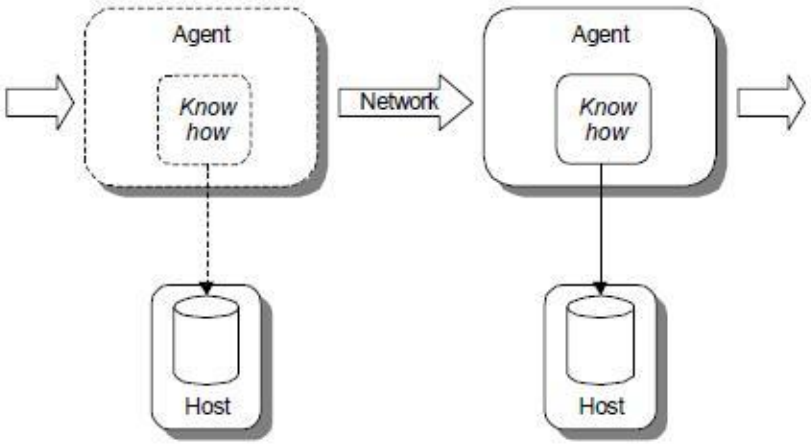

Figure 2.5 Mobile Agent Paradigm [44]

In mobile agent paradigm, the know-how (code) of doing a process is not tied to any host. Here the client and server are merged to become host and provide high degree of flexibility. The mobile agents contain the code and it move between the hosts to provide the know-how. As compared to traditional systems where data is sent to the code, in mobile agent paradigm the code is sent to the data.

Software mobile agents have many advantages over traditional system. These advantages are discussed in the next section.

2.2.2 Advantages of Software Mobile Agents

The followings are the advantages of using software mobile agents [44] [45]:

- Reduce the network load: mobile agent contains the know-how, so the interactions take place locally. This reduces the flow of raw data over network.
- Overcome network latency: mobile agents dispatched from a central controller and act locally and directly execute the controller's directions.
- Encapsulate protocols: in distributed system, each host owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data. When protocols evolve to use new needs it's a cumbersome task to upgrade protocol code properly. On the other hand, mobile agents create "channels" based on proprietary protocols.
- Execute asynchronously and autonomously: tasks are embedded into mobile agents. Mobile agents are independent of the creating process, and act asynchronously and autonomously.
- Adapt dynamically: mobile agents can sense the changes in the environment and can act accordingly.
- Naturally heterogeneous: mobile agents are computer- and transport layer independent. They just rely on their execution environment and provide seamless system integrity.
- Robust and fault-tolerant: mobile agents can react dynamically to unfavorable changes in the environment which helps to create a robust and fault-tolerant grid system.

Mobile agents differ largely from traditional system in architecture and implementation. So, mobile agent computing paradigm needs new standards. Next subsection discusses about available standards.

2.2.3 Mobile Agent Standardization

There are two standards available for the mobile agents [46]. These are:

- Mobile Agent System Interoperability Facility (MASIF): This standard is developed by Object Management Group (OMG). MASIF addresses interfaces

between agent systems. MASIF does not provide interoperability of coding languages. Systems written in same coding language are interoperable. MASIF standardizes the following four areas: agent management, agent transfer, agent and agent system names, and agent system type and location syntax.

- Foundation for Intelligent Physical Agents (FIPA): FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. It provides standards for agent communication, agent transport, agent management, and abstract architecture and application.

2.2.4 Mobile Agent Systems

Mobile agent based computing is researched through past year and developing dynamically and fast. Majority of mobile agent systems are based on java. Some popular java based mobile software agent systems are Aglets, Odyssey, Concordia and Voyager. The most popular system is Aglets brings the flavor of mobility in java applet. Some non-java based mobile agent systems are Agent Tcl, Ara, and TACOMA.

2.2.5 Mobile Agent based Monitoring systems

Mobile agents can be a good choice for building monitoring system for grid computing. They can easily cope up with the heterogeneity. The network bandwidth and latency can be saved by using mobile agents. As mobile agents can travel through the network, they can easily gather the information about the grid nodes. The information of grid nodes can easily be generated using APIs locally. Then these data can be processed and display to the users using an interface.

MAP [43] is a software mobile agent oriented monitoring system. In MAP three different levels of mobile agents is used, Low-Level Sensors, Sensor Agents, and High-Level Monitoring.

Conclusion

In this chapter, grid monitoring systems have been discussed in detail and different available monitoring systems are described according to their taxonomy and their comparisons have also been reported. Different kind of faults which occurs in grid and different procedures to mitigate these faults have also been discussed. Lastly in

this chapter mobile agents and their use in monitoring systems has been discussed briefly. Next chapter discusses the problem formulation of the thesis work.

Chapter 3 Problem Formulation

Chapter 2 discussed about different monitoring systems available and reported their comparison on the basis of GMA mapping, implementation, key features and security. The deployment and maintenance of these monitoring systems are difficult and tricky. Users needed to have deep technical knowledge to setup and run these monitoring systems.

To resolve these complexities mobile agents can be used to develop a monitoring system for grid environment. Mobile agents can move freely between the nodes of grid and collect the necessary information needed from nodes.

3.1 Objectives

The objectives of proposed systems are:

- Create a mobile agent based monitoring system that is easy to deploy and manage.
- Monitoring system should be scalable.
- Monitoring data should be easily accessible to the users.
- Monitoring system should be flexible such that any node can act as the server node in case of failure.

3.2 Methodology

The objectives can be achieved in the following way:

- Resource information will be generated locally on the nodes and stored in files.
- Mobile agents will travel between the server node and other nodes. Mobile agents collect resource data from the file and comeback to the server node.
- Data will be updated to database on the server node.
- Web pages will be created to display monitored data.
- Web pages will be hosted over the local network so that other nodes of the network can access the site.
- Any node that manages and sends the mobile agents can become the server so any failure in the server node has minimum effect on the monitoring system.

The next chapter discusses the implementation details of the proposed monitoring system.

Chapter 4 Design and Implementation of the Proposed System

The proposed mobile agent based monitoring system is designed using various open source technologies. This chapter describes all these technologies and discusses how they are used. This chapter also provides details of the design and implementation of the proposed monitoring system.

4.1 Design of the Proposed Monitoring System

This section discusses about the design of monitoring system. It describes the topology and architecture of the proposed monitoring system and further the design of agents and database tables.

4.1.1 Topology of the Monitoring System

The topology of the monitoring system is shown below:

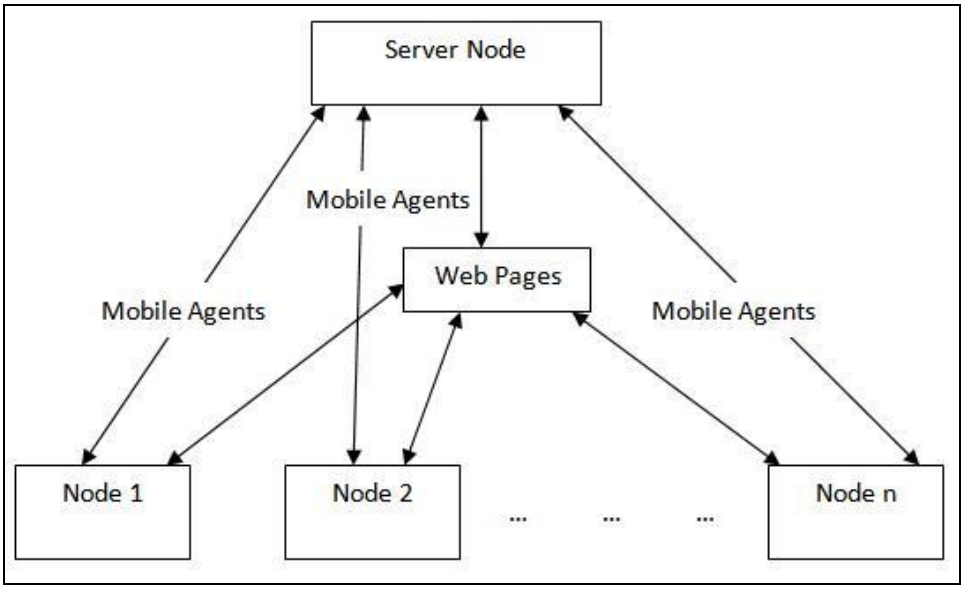

Figure 4.1 Topology of the Mobile Agent based Monitoring System

Server node sends mobile agents to the other nodes of the network. These mobile agents gather resource information from the nodes. Then they return to the server node and update the database. Web pages to provide these resources information to users are created and hosted on local network using apache HTTP server [47]. Any system on local network running apache HTTP server can access these web pages.

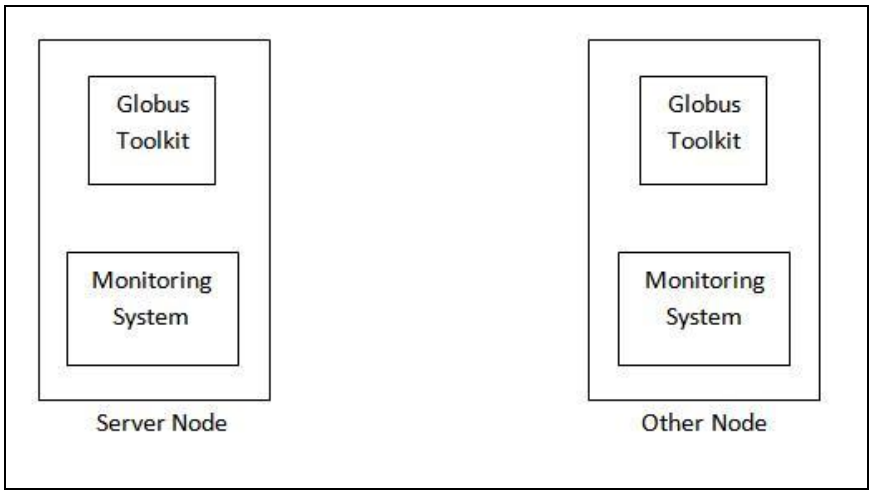

Figure 4.2 Nodes and Grid Computing

The monitoring system is proposed to monitor systems used in grid computing. Figure 4.2 shows how the nodes are related to grid computing. Every nodes participating in the monitoring system is part of grid and the grid is created using globus. Figure 4.2 shows that the nodes have Globus Toolkit installed.

4.1.2 Architecture of the Monitoring System

This section discusses about the architecture of mobile agent based monitoring system. Figure 4.3 presents a pictorial view of the system.

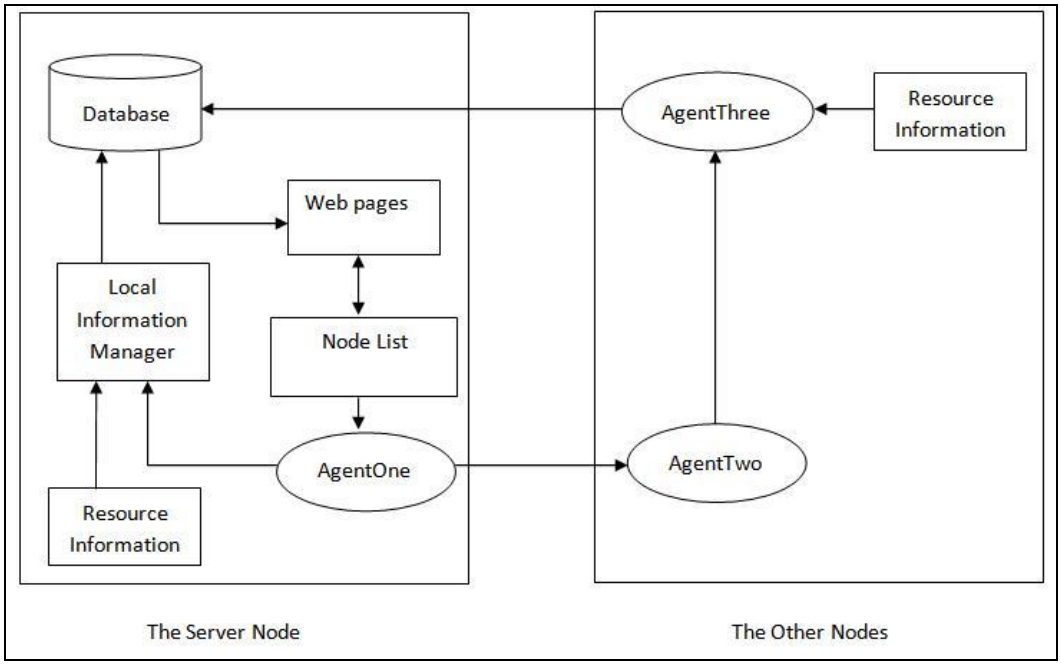

Figure 4.3 Architecture of Mobile Agent based Monitoring System

The system uses Hyperic SIGAR [48] to gather information about the resources of local system. This resource information is managed, processed and updated to a database. Then this information is accessed through web pages and presented to the users. The web pages can be accessed from any nodes from the network. The local resource information and remote system's resource information is collected differently. Local Information Manager is used to collect and process the local resource information and updates the database, and mobile agents are used to collect and process the resource information of remote nodes. To manage resource information of remote systems three mobile agents are used. These mobile agents are: AgentOne, AgentTwo, and AgentThree. AgentOne invokes the local information manager.

Agents are needed to dispatch to the remote nodes that are needed to be monitored. Addresses of these nodes are stored in a file which is denoted as "Node List" in the Figure 4.3. AgentOne gets the addresses of remote nodes from the list, and then it creates and sends the AgentTwo to the remote hosts. On the remote host, AgentTwo creates AgentThree. AgentThree gathers local resources information of the remote node. Then AgentTwo sends AgentThree back to the server node. On the server node, AgentThree update the database with resource information. Web pages designed and deployed on the server node to access the resource information from the database and represent these data to the users.

4.1.3 Design of Mobile Agents

Aglets-2.0.2 [49] used in the monitoring system. Mobile agents are dispatched from the server node to remote nodes, after gathering information these mobile agents carries back the information to the server node and update the database. Three mobile agents are used in the monitoring system as already discussed. These agents are AgentOne, AgentTwo, and AgentThree. Users need to feed AgentOne to the Tahiti Server; users also need to dispose the AgentOne. AgentTwo and AgentThree are generated and disposed automatically. AgentOne creates AgentTwo and sends AgentTwo to remote nodes. AgentTwo creates AgentThree on remote host and sends back AgentThree to server node.

Next subsections give detailed description of working of AgentOne, AgentTwo and AgentThree. These mobile agents are discussed using work flow diagrams.

a) Design of AgentOne

AgentOne starts the monitoring process. Figure 4.4 shows the workflow of AgentOne.

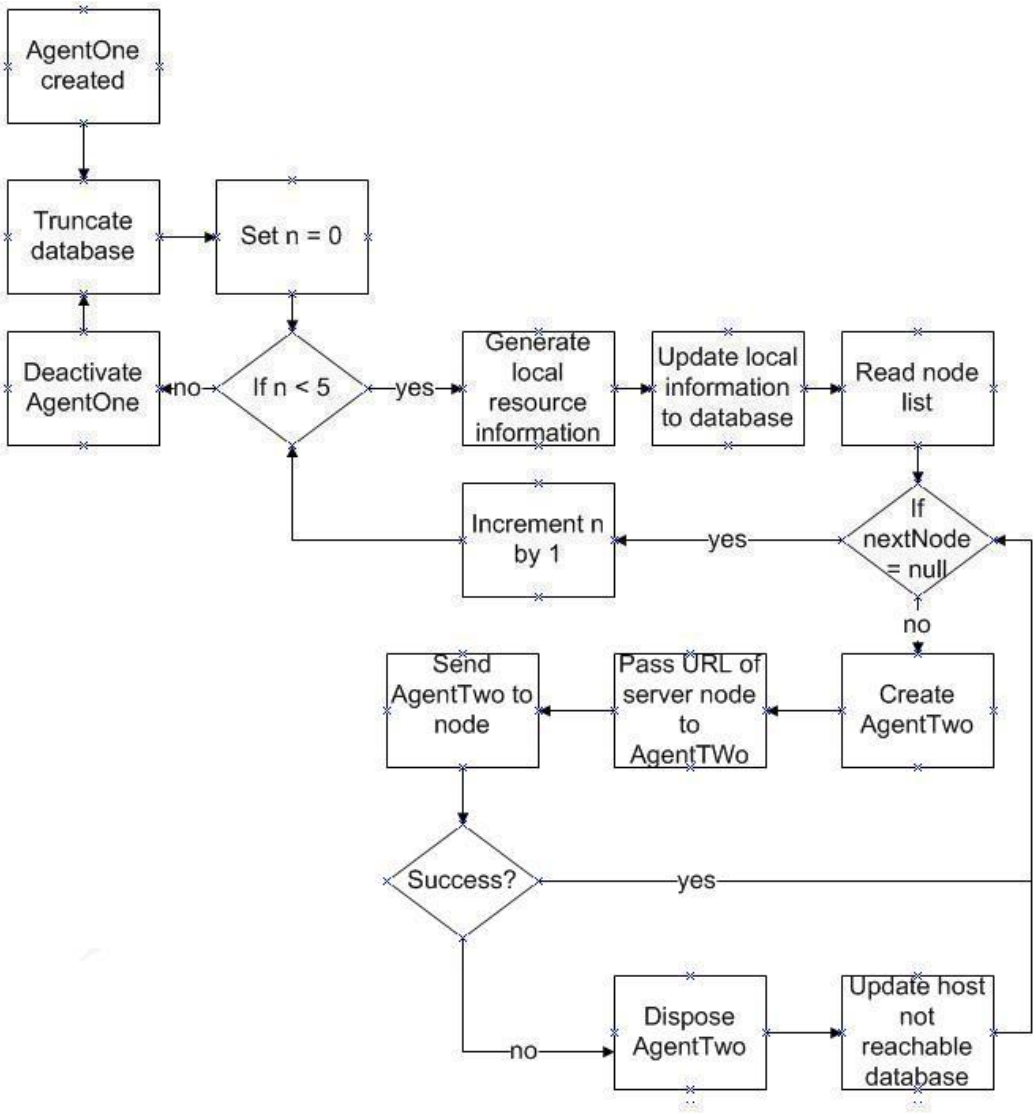

Figure 4.4 Workflow of AgentOne

AgentOne truncates the monitoring systems database to delete all the previous values stored in the database. AgentOne calls Hyperic SIGAR API to generate system information and local information manager to update the information to the database. Then it reads the node list to find out remote hosts addresses. AgentOne creates AgentTwo and sends AgentTwo to these addresses. AgentTwo needs the address of server node as it sends back AgentThree from remote nodes. So AgentOne sends the URL of server to the AgentTwo. If sending of AgentTwo to an address fails,

AgentOne disposes that AgentTwo. That address is declared unreachable and updated to database. These processes go on in a loop until user stops it by disposing AgentOne.

b) Design of AgentTwo

Figure 4.5 shows the workflow of AgentTwo:

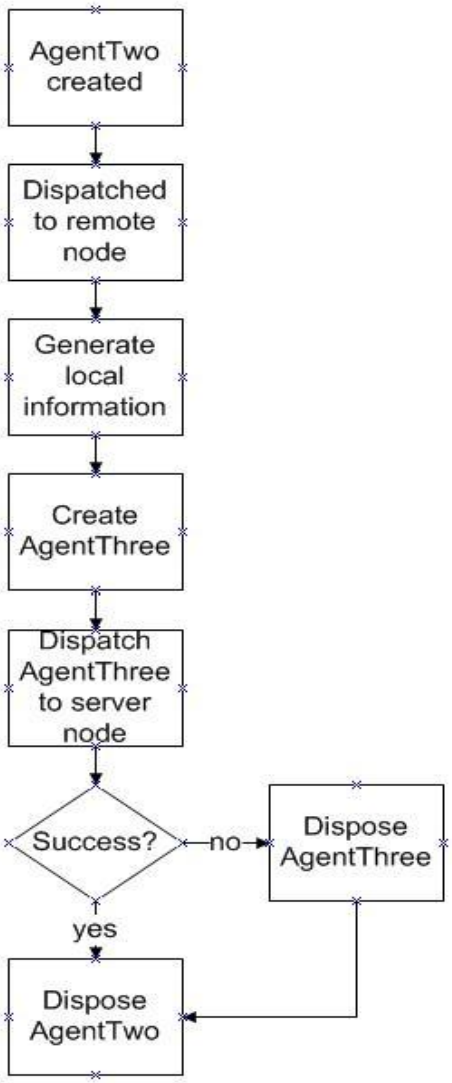

Figure 4.5 Workflow of AgentTwo

Though AgentTwo starts its life in server node its original working is done in remote node. AgentOne creates AgentTwo on server node and sends AgentTwo to the addresses found on node list. Before sending AgentTwo, AgentOne passes address of server node to AgentTwo. On remote node AgentTwo calls the Hyperic SIGAR API to generate resource information. Then it creates AgentThree and sends it to server

node. If dispatching of AgentThree to server fails, AgentTwo disposes AgentThree. At last AgentTwo disposes itself.

c) Design of AgentThree

Figure 4.6 shows the workflow of AgentThree:

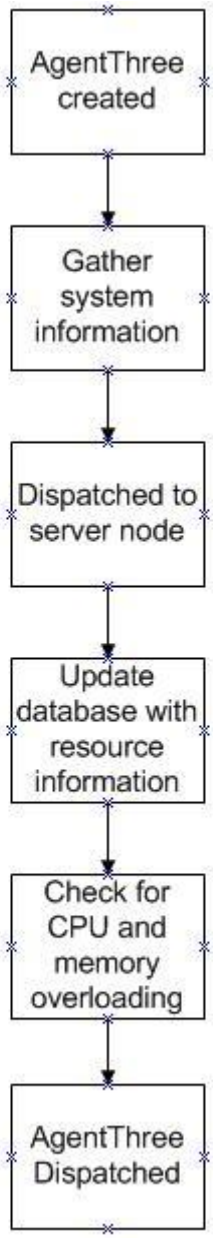

Figure 4.6 Work flow of AgentThree

AgentThree is created by AgentTwo on remote node. AgentThree then gather system information. AgentThree reads the files created by Hyperic SIGAR and initialize its variables by the values it gets from those files. Then AgentThree is dispatched to the server node. On server node AgentThree updates the database. It

also checks if any remote host has idle CPU and used memory size below threshold value. The threshold value is defined in the AgentThree. If overloading found then an error is updated in database. At last AgentThree disposes itself.

4.1.4 Design and Schema of Database Tables

MySQL 5.0.45 [50] is used to create and manage database. A database named monitor is created. Figure 4.7 shows the tables of the monitor database.

```
[root@nodea ~]# mysql -u mydbadmin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.0.45 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use monitor;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_monitor |
+------------------+
| cpuInfo          |
| errorInfo        |
| hostReach        |
| memoryInfo       |
| networkInfo      |
| osInfo           |
| systemInfo       |
+------------------+
7 rows in set (0.00 sec)
```

Figure 4.7 Tables of monitor Database

monitor database contains seven tables. They are cpuInfo, errorInfo, hostReach, memoryInfo, networkInfo, osInfo and systemInfo. IP address of nodes is used as primary key in all tables.

a) Details of cpuInfo table

cpuInfo table is used to store CPU related information of nodes. Figure 4.8 shows the schema of the table.

```
mysql> describe cpuInfo;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ipAddress | varchar(20) | NO   | PRI |         |       |
| cpuVendor | varchar(20) | YES  |     | NULL    |       |
| cpuModel  | varchar(20) | YES  |     | NULL    |       |
| cpuSpeed  | varchar(20) | YES  |     | NULL    |       |
| totalCpu  | varchar(20) | YES  |     | NULL    |       |
| cacheSize | varchar(20) | YES  |     | NULL    |       |
| cpuIdle   | varchar(20) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

Figure 4.8 Schema of cpuInfo Table

b) Details of errorInfo table

AgentThree updates this table if it founds any overloading in CPU and/or memory usage. The schema of errorInfo table is shown below.

```
mysql> describe errorInfo;
+-----------+-------------+------+-----+-------------------+-------+
| Field     | Type        | Null | Key | Default           | Extra |
+-----------+-------------+------+-----+-------------------+-------+
| ipAddress | varchar(20) | YES  |     | NULL              |       |
| hostName  | varchar(20) | YES  |     | NULL              |       |
| time      | timestamp   | NO   |     | CURRENT_TIMESTAMP |       |
| error     | varchar(35) | YES  |     | NULL              |       |
+-----------+-------------+------+-----+-------------------+-------+
4 rows in set (0.00 sec)
```

Figure 4.9 Schema of errorInfo Table

time is automatic and it captures the current time when it is updated. error contains the type of error and the percentage of CPU or memory is being used.

c) Details of hostReach table

This is updated by AgentOne when sending of AgentTwo to any address failed. This table is used to determine the unreachable hosts on network.

```
mysql> describe hostReach;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ipAddress | varchar(20) | NO   | PRI |         |       |
| status    | varchar(10) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Figure 4.10 Schema of hostReach Table

d) Details of memoryInfo table

memoryInfo table stores information about memory. Figure 4.11 shows the schema:

```
mysql> describe memoryInfo;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| ipAddress   | varchar(20) | NO   | PRI |         |       |
| totalMemory | varchar(20) | YES  |     | NULL    |       |
| usedMemory  | varchar(20) | YES  |     | NULL    |       |
| freeMemory  | varchar(20) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Figure 4.11 Schema of memoryInfo Table

e) Details of networkInfo

All the networking related information of nodes is stored in this table. The schema is shown below:

```
mysql> describe networkInfo;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| ipAddress        | varchar(20) | NO   | PRI |         |       |
| primaryInterface | varchar(20) | YES  |     | NULL    |       |
| MACAddress       | varchar(20) | YES  |     | NULL    |       |
| netMask          | varchar(20) | YES  |     | NULL    |       |
| domainName       | varchar(20) | YES  |     | NULL    |       |
| gateWays         | varchar(20) | YES  |     | NULL    |       |
| dns              | varchar(20) | YES  |     | NULL    |       |
| secondDns        | varchar(20) | YES  |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
8 rows in set (0.01 sec)
```

Figure 4.12 Schema of networkInfo Table

f) Details of osInfo

OS related information is stored in this table.

```
mysql> describe osInfo;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| ipAddress     | varchar(20) | NO   | PRI |         |       |
| osDescription | varchar(20) | YES  |     | NULL    |       |
| osName        | varchar(20) | YES  |     | NULL    |       |
| osArch        | varchar(20) | YES  |     | NULL    |       |
| osMachine     | varchar(20) | YES  |     | NULL    |       |
| osVersion     | varchar(20) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Figure 4.13 Schema of osInfo Table

g) Details of systemInfo

In systemInfo table stores data about up time, load average, user logged and globus status. globusRunning is set yes if globus found running else it set no. It also store hostname and system time. The schema of this table is shown below:

```
mysql> describe systemInfo;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| ipAddress     | varchar(20) | NO   | PRI |         |       |
| hostName      | varchar(20) | YES  |     | NULL    |       |
| systemTime    | varchar(20) | YES  |     | NULL    |       |
| upTime        | varchar(20) | YES  |     | NULL    |       |
| loadAverage   | varchar(20) | YES  |     | NULL    |       |
| loggedUser    | varchar(20) | YES  |     | NULL    |       |
| globusRunning | varchar(50) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

Figure 4.14 Schema of systemInfo Table

4.1.5 Design of Web Pages

Web pages are created using PHP. These web pages retrieve data from MySQL database. The following Figure 4.15 shows the schema of the web pages.

```
<?xml version="1.0" encoding="UTF-8"?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
 <siteMapNode title="" description="" url="~/">
  <siteMapNode title="addnode" description="addnode" url="~/addnode.php"/>
  <siteMapNode title="detail php?ipAddress=172 31" description="detail php?ipAddress=172 31" url="~/detail.php?ipAddress=172.31.5.153"/>
  <siteMapNode title="detail php?ipAddress=172 31" description="detail php?ipAddress=172 31" url="~/detail.php?ipAddress=172.31.5.180"/>
  <siteMapNode title="detail php?ipAddress=172 31" description="detail php?ipAddress=172 31" url="~/detail.php?ipAddress=172.31.5.187"/>
  <siteMapNode title="error php?ipAddress=172 31" description="error php?ipAddress=172 31" url="~/error.php?ipAddress=172.31.5.153"/>
  <siteMapNode title="error php?ipAddress=172 31" description="error php?ipAddress=172 31" url="~/error.php?ipAddress=172.31.5.180"/>
  <siteMapNode title="error php?ipAddress=172 31" description="error php?ipAddress=172 31" url="~/error.php?ipAddress=172.31.5.187"/>
  <siteMapNode title="index" description="index" url="~/index.php"/>
  <siteMapNode title="http:" description="http:" url="~/http:/">
  </siteMapNode>
 </siteMapNode>
</siteMap>
```

Figure 4.15 Sitemap of Web Pages

The index.php shows general information about monitored nodes. It also shows the list of unreachable nodes. index.php links to detail.php, addnode.php and error.php. detail.php shows all information of node. Using addnode.php user can add new nodes for monitoring and can remove nodes. error.php shows the detail of CPU and memory overloading.

The web pages are hosted on the local network using apache HTTP web server. Any system on local network can access these web pages.

4.2 Implementation Details

This section discusses about the implementation details of the agent based monitoring system. Fedora Core 8 is used as operating system in implementation.

4.2.1 Technologies Used

In the proposed system java based mobile agent framework aglets is used. Hyperic SIGAR is used to gather system information locally. MySQL database system is used to maintain the monitoring data. The website is designed using PHP and hosted on local network using apache HTTP server.

a) Globus Toolkit

Globus Toolkit 4.2.1 [11] is used to create grid for deploying and testing the proposed monitoring system. Globus Toolkit is developed and provided by Globus Alliance. Globus Toolkit includes software and libraries for resource monitoring, resource management, data management, communication, fault detection, security and portability. GT4 uses Web services mechanism heavily to provide flexible, extensible, and widely adopted XML-based mechanisms for describing, discovering, and invoking network services. Globus Toolkit is open source, well documented and mailing-list is available for users query.

b) Hyperic SIGAR

Hyperic System Information Gatherer (SIGAR) [48] is an open source cross-platform API. SIGAR supports Linux, Windows, FreeBSD, Solaris etc operating system and architecture. SIGAR API gives portable access to inventory and monitoring data including:

- System memory, swap, CPU, load average, uptime, logins.
- Per-process memory, CPU, credential info, state, arguments, environment, open files.
- File system detection and metrics.
- Network interface detection, configuration information and metrics.
- Network route and connection tables.

Hyperic SIGAR is operated from the command-line tools. Hyperic SIGAR has been used in implementation as it is easy to use, open source and well documented.

c) Aglets

Aglets [49] [46] is a java based mobile agent platform. An aglet is a java mobile agent which can autonomously and spontaneously move from a system to another. An aglet is simply an object on which a thread executes on. This approach is very similar to the applets or servlets. Aglets has been used in the proposed monitoring system as it is open source and well documented. Aglets can migrate in both directions (in and out the platform) and message between local and migrated agents. Aglets supports weak mobility, only code is needed to transport without any particular information about the execution of the code, and aglets mobile agents are restarted from the entry-point of the code after migration to other machines. To overcome these limitation aglets is designed to keep its java state after migration. The aglets code will restart from a entry point but no re-initialization will happen. Proxy is used in aglets for communication without hampering the security.

Aglets mobile agent platform is composed of following three parts:

- Aglet mobile agent platform: is the core platform, able to manage the agents.
- Tahiti: the server in-charge of managing the mobility of the agents. It comes with a graphical user interface.
- Aglets software development kit: is a library that provides developers all the facilities required to write mobile agents complaint to the Aglets MAP.

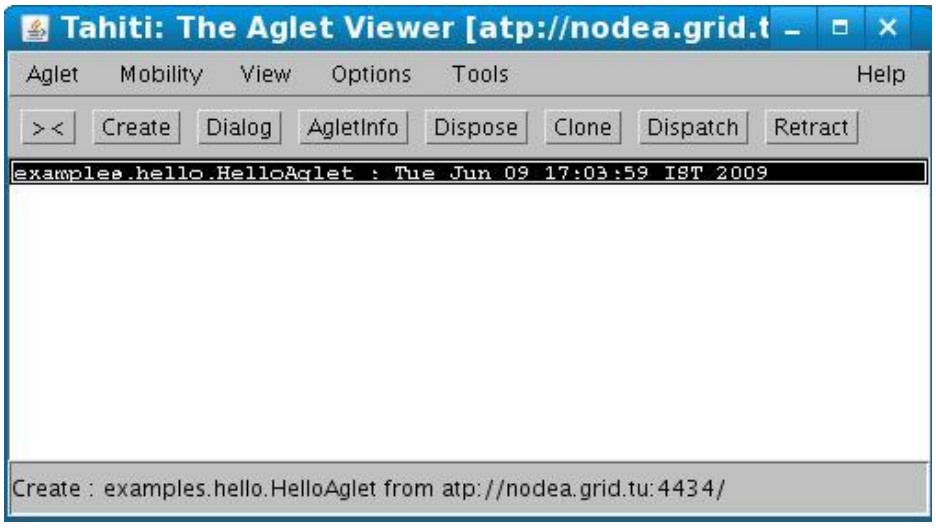

Figure 4.16 Tahiti Server

Figure 4.16 shows the Tahiti server, it provides functions to manage aglets mobile agents. The basic functions are [51]:

- Create: allow administrators to create new agent instances.

- Dialog: sends message to the selected agent.
- AgletInfo: opens dialog window with information about the selected agent.
- Dispose: allow administrator to kill a running agent.
- Clone: allow administrator to create an identical copy of running agent.
- Dispatch: allow administrator to order an agent to migrate to another aglets platform.
- Retract: allow administrator to order an agent to come back from a remote aglets platform.

Aglets has been used in the implementation for its ease of use, functionalities and documentation.

d) Java

Java [52] is an object oriented programming language. Java is used to manage the local information data of the server. Java is used to gather information from the Hyperic SIGAR and update the information to the database. Java is ideal choice for programming grid applications as it is architectural neutral. So java code written code written for one architecture can run on other architectures and no change is required in the code as such. Hence java is used in the implementation.

e) MySQL

MySQL [50] is used as the backend. MySQL is multithreaded, multi-user, SQL database management system. MySQL is the most used open source database. MySQL has become preferred choice for web based development for its speed, reliability and ease of use and hence MySQL is used in the implementation.

f) PHP

PHP [53] is used for creating the front end (web pages) of the monitoring system. PHP is a widely used general-purpose language for web development. PHP is a powerful tool for making dynamic and powerful web pages. PHP's principle focus is on server-side scripting. PHP is used embedded with HTML. PHP generally runs on a web server. It can be deployed using almost any web server on any platform. PHP can connect with MySQL easily hence has been used for implementation.

g) Apache HTTP server

Apache HTTP server [47] is used to process and host the PHP web pages. Apache HTTP server takes PHP codes as input and creates web pages as output. Apache is

open source and available for variety of operating systems. Apache has been used for hosting web pages on local network using Virtual Host function.

The combination of Apache HTTP server, MySQL, and PHP with Linux operating system creates a software bundle named LAMP. This software bundle is very popular among developers because of its low acquisition cost and ubiquity of its components and hence has been used.

4.2.2 Resource Information Generation

Hyperic SIGAR 1.6.2 is used for resource information generation.

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar help
Available commands:
        alias           - Create alias command
        cpuinfo         - Display cpu information
        df              - Report filesystem disk space usage
        du              - Display usage for a directory recursively
        free            - Display information about free and used memory
        get             - Get system properties
        help            - Gives help on shell commands
        ifconfig        - Network interface information
        iostat          - Report filesystem disk i/o
        kill            - Send signal to a process
        ls              - simple FileInfo test at the moment (like ls -l)
        mps             - Show multi process status
        netinfo         - Display network info
        netstat         - Display network connections
        nfsstat         - Display nfs stats
        pargs           - Show process command line arguments
        penv            - Show process environment
        pfile           - Display process file info
        pidof           - Find the process ID of a running program
        pinfo           - Display all process info
        pmodules        - Display process module info
        ps              - Show process status
        quit            - Terminate the shell
        route           - Kernel IP routing table
        set             - Set system properties
        sleep           - Delay execution for the a number of seconds
        source          - Read a file, executing the contents
        sysinfo         - Display system information
        time            - Time command
        ulimit          - Display system resource limits
        uptime          - Display how long the system has been running
        version         - Display sigar and system version info
        who             - Show who is logged on
```

Figure 4.17 Commands Available in Hyperic SIGAR

Figure 4.17 shows all the commands available in Hyperic SIGAR 1.6.2. In the monitoring system cpuinfo, free, ps, sysinfo, netinfo, uptime, and who is used to gather information about CPU, memory, network etc.

These commands produce raw data and information needed to be extracted from these output. To do the extraction shell script files are created for every command. These shell scripts automatically call these Hyperic SIGAR command and extract the needed data. A master shell script file is created to call all these script files and generate a file containing all information. Mobile agents access this file to obtain information about the system.

a) Globus related Information Generation

This monitoring system is intended to use in grid computing. So it is necessary to generate information about globus. This monitoring system checks whether globus is running on the target system or not. To check this, ps command of Hyperic SIGAR is used. Figure 4.18 shows partial output of the ps command.

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar ps
1       root    10:49   2.1M    696K    604K    S       0:0     init [5]
2       root    10:49     0       0       0     S       0:0     kthreadd
3       root    10:49     0       0       0     S       0:0     migration/0
4       root    10:49     0       0       0     S       0:0     ksoftirqd/0
5       root    10:49     0       0       0     S       0:0     watchdog/0
6       root    10:49     0       0       0     S       0:0     events/0
7       root    10:49     0       0       0     S       0:0     khelper
58      root    10:49     0       0       0     S       0:0     kblockd/0
61      root    10:49     0       0       0     S       0:0     kacpid
62      root    10:49     0       0       0     S       0:0     kacpi_notify
192     root    10:49     0       0       0     S       0:0     cqueue/0
194     root    10:49     0       0       0     S       0:0     ksuspend_usbd
199     root    10:49     0       0       0     S       0:0     khubd
202     root    10:49     0       0       0     S       0:0     kseriod
237     root    10:49     0       0       0     S       0:0     pdflush
238     root    10:49     0       0       0     S       0:0     pdflush
239     root    10:49     0       0       0     S       0:0     kswapd0
291     root    10:49     0       0       0     S       0:0     aio/0
383     root    10:49     0       0       0     S       0:0     khvcd
446     root    10:49     0       0       0     S       0:0     kpsmoused
472     root    10:49     0       0       0     S       0:0     ata/0
473     root    10:49     0       0       0     S       0:0     ata_aux
476     root    10:49     0       0       0     S       0:0     scsi_eh_0
477     root    10:49     0       0       0     S       0:0     scsi_eh_1
478     root    10:49     0       0       0     S       0:0     scsi_eh_2
479     root    10:49     0       0       0     S       0:0     scsi_eh_3
```

Figure 4.18 Partial Output of the ps Command

Following output shows all the running processes on the system. The process of globus is "java:org.globus.bootstrap.ContainerBootstrap". Shell script is written to look for this specific entry in the ps output. Figure 4.19 shows the globus process in output of ps command.

```
2747    globus  15:25   4.6M    1.4M    1.2M    S       0:0     -bash
2792    globus  15:25   663M     67M    7.5M    S       0:12    java:org.globus.bootstrap.ContainerBootstrap
2814    globus  15:25   5.9M    1.8M    1.5M    S       0:0     /opt/globus-4.2.1/libexec/globus-scheduler-event-generator
2896    root    15:26   4.7M    1.5M    1.2M    S       0:0     bash
2914    root    15:26   208M     15M    7.6M    S       0:0     java:org.hyperic.sigar.cmd.Runner
```

Figure 4.19 Globus Process in ps Command Output

b) Memory Information Generation

Free command of Hyperic SIGAR is used to generate information regarding memory information. Figure 4.20 shows the output:

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar free
             total       used       free
Mem:       1033832     383672     650160
-/+ buffers/cache:     171272     862560
Swap:      1044216          0    1044216
RAM:         1024MB
```

Figure 4.20 Output of the free Command

The first line of this output is extracted using a shell script file.

c) CPU information generation

The output of cpuinfo command of Hyperic SIGAR is shown below:

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar cpuinfo
Vendor.........AMD
Model..........Opteron
Mhz............2200
Total CPUs.....1
Cache size....1024

CPU 0.........
User Time.....0.0%
Sys Time......0.0%
Idle Time.....100.0%
Wait Time.....0.0%
Nice Time.....0.0%
Combined......0.0%
Irq Time......0.0%
SoftIrq Time..0.0%
Stolen Time....0.0%

Totals........
User Time.....15.6%
Sys Time......1.9%
Idle Time.....82.3%
Wait Time.....0.0%
Nice Time.....0.0%
Combined......17.6%
Irq Time......0.0%
SoftIrq Time..0.0%
Stolen Time....0.0%
```

Figure 4.21 Output of cpuinfo Command

The information regarding vendor, model, MHz (speed), total CPUs, Cache size and Idle time are useful and needed to be extracted.

d) Network Information Generation

To generate information regarding network the netinfo command of Hyperic SIGAR is used. The output is shown below:

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar netinfo
primary interface.....eth0
primary ip address....172.31.5.187
primary mac address...00:E0:81:5B:AC:1B
primary netmask.......255.255.255.0
host name.............nodea.grid.tu
domain name...........(none)
default gateway.......172.31.5.1
primary dns...........172.31.1.6
secondary dns.........
```

Figure 4.22 Output of the netinfo Command

All information generated by this command is useful and shell script is written to extract information.

e) System Information Generation

System information is generated using the sysinfo command of Hyperic SIGAR.

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar sysinfo
Sigar version.......java=1.6.2.0, native=1.6.2.0
Build date..........java=05/14/2009 03:00 PM, native=05/14/2009 03:00 PM
SCM rev.............java=exported, native=exported
Archlib.............libsigar-x86-linux.so
Current fqdn........nodea.grid.tu
Current user........root

OS description......Fedora 8
OS name.............Linux
OS arch.............i686
OS machine..........i686
OS version..........2.6.23.1-42.fc8
OS patch level......unknown
OS vendor...........Fedora
OS vendor version...8
OS code name........
OS data model.......32
OS cpu endian.......little
Java vm version.....11.3-b02
Java vm vendor......Sun Microsystems Inc.
Java home...........//opt/jdk1.6.0_13/jre

  11:09 AM  up 20 min, load average: 0.29, 0.12, 0.09

Vendor.........AMD
Model..........Opteron
Mhz............2200
Total CPUs.....1
Cache size....1024


             total       used       free
Mem:       1033832     387448     646384
-/+ buffers/cache:     173216     860616
Swap:      1044216          0    1044216
RAM:        1024MB

File Systems.........[/, /proc, /sys, /dev/pts, /dev/shm, /proc/sys/fs/binfmt_misc, /var/lib/nfs/rpc_pipefs, /media/_1]

Network Interfaces...[lo, eth0]

System resource limits:
core file size.......0
data seg size........unlimited
file size............unlimited
pipe size............8
max memory size......unlimited
```

Figure 4.23 Output of the sysinfo Command

Only OS specific information is extracted using shell script. Other information is also useful but gathered using other commands.

To obtain information about uptime and load average, uptime command of Hyperic SIGAR is used and the information extracted by a shell script file.

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar uptime
  11:10 AM  up 21 min, load average: 0.14, 0.12, 0.09
```

Figure 4.24 Output of the uptime Command

Figure 4.24 shows the output of the uptime command. The command outputs the current time of the system. As grid needs all participating node's clock to be synchronized, so the current time is not needed to be extracted.

Load average more or less represents the average number of processes that are in the running (using the CPU) or runnable (waiting for the CPU) states. The load

average is calculated as an exponential moving average of the load number (the number of processes that are running or runnable). The three numbers returned as the system's load average represent the one, five, and fifteen minute moving load average of the system.

Information about the user who is logged in the system is also necessary. who command of Hyperic SIGAR gives this information. Figure 4.25 shows the output of the command.

```
[root@nodea ~]# java -jar /opt/hyperic-sigar-1.6.2-src/bindings/java/sigar-bin/lib/sigar.jar who
root     tty7     Jun 09 10:50    (:0)
root     pts/0    Jun 09 10:51    (:0.0)
root     pts/1    Jun 09 10:57    (:0.0)
```

Figure 4.25 Output of the who Command

Conclusion

This chapter gives detailed description of design of the monitoring system. It lists all the used technologies and how they are used. Work flow of all three mobile agents are shown and discussed. Then generation of system information and design of MySQL and web pages are also discussed. Next chapter shows the experimental results of the proposed monitoring system.

Chapter 5 Experimental Results

This chapter details the deployment and working of the monitoring system.

5.1 Deployment Details

The monitoring system is deployed in the Research Lab of Computer Science and Engineering Department of Thapar University. Five systems are used in deployment. The operating system of these systems is Fedora Core 8. Globus is installed in all those system. To deploy the monitoring system aglets and Hyperic SIGAR are needed to be installed in those systems. These two are installed in the /opt folder. The shell script files are stored in a folder named Monitor. The folder is copied to /opt/aglets/cnf. This folder also contains the Java file to process local system information i.e. local information manager. The agent codes are copied to /opt/aglets/public folder. All these codes are compiled. In the server node, /opt/aglets/cnf folder contains a file named ip.txt which contains the addresses of the nodes. The owner of this ip.txt file needed to be changed to apache.

5.2 Creation and Deployment of Agents

AgentOne is created using Tahiti, further AgentTwo and AgentThree are created automatically.

5.2.1 Adding AgentOne in Tahiti

In the server node, to add AgentOne in the Tahiti, “Create” button of Tahiti is used. Clicking the Create will result in a pop up screen named “Create Aglet” (Figure 5.1). In the Aglet name field AgentOne is entered. Then Add to List button is needed to be pressed to add this aglet to the aglets list. Then pressing the create button will create the AgentOne aglet. If the Java codes of agents is in the /opt/aglets/public folder then simply typing the aglets name will work. But if the codes are in subfolders of /opt/aglets/public, that the codes are under some package, then packagename.codename will work.

5.2.2 Agent Life Cycle

After adding AgentOne, AgentOne will automatically create AgentTwo and sends it to the remote node. On remote node AgentTwo automatically generates AgentThree and sends it to the server node. These agents work in three stages.

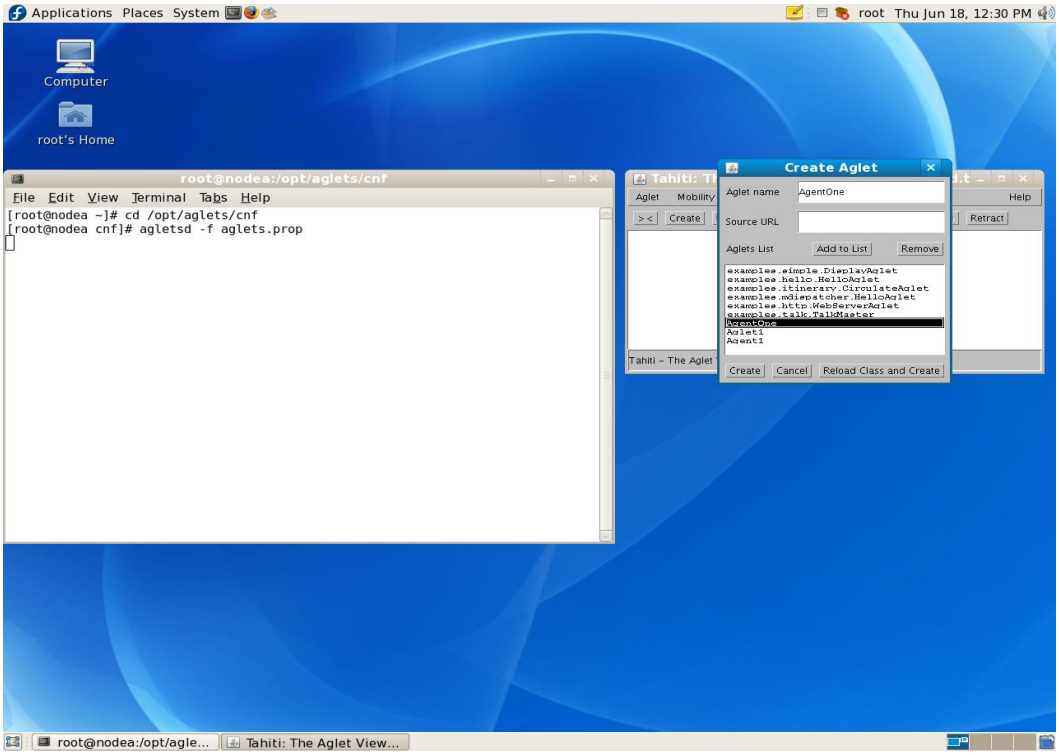

Figure 5.1 Adding AgentOne in Tahiti

Stage One

The monitoring process begins with adding AgentOne in Tahiti server on AgentOne. Figure 5.2 shows AgentOne running on Tahiti in server node.

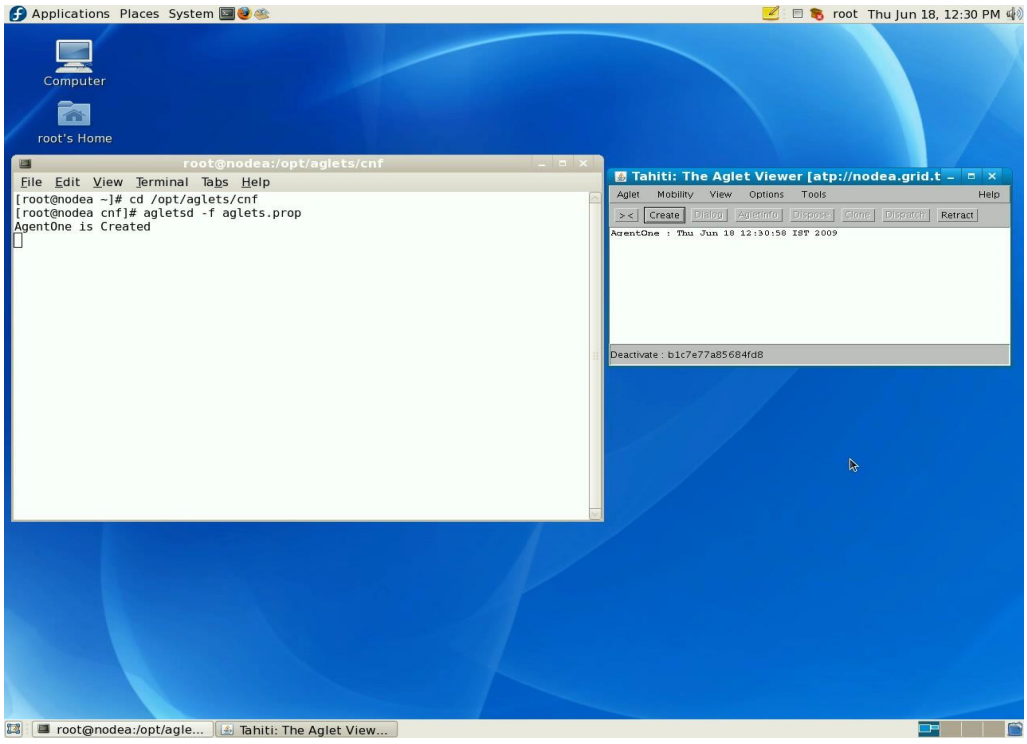

Figure 5.2 AgentOne Running on Server

AgentOne calls local information manager to create local system information and update the information to database. AgentOne runs in loop. It continuously reads the node list and sends AgentTwo to these addresses. AgentOne creates AgentTwo (Figure 5.3). If sending of AgentTwo fails, AgentOne disposes AgentTwo.

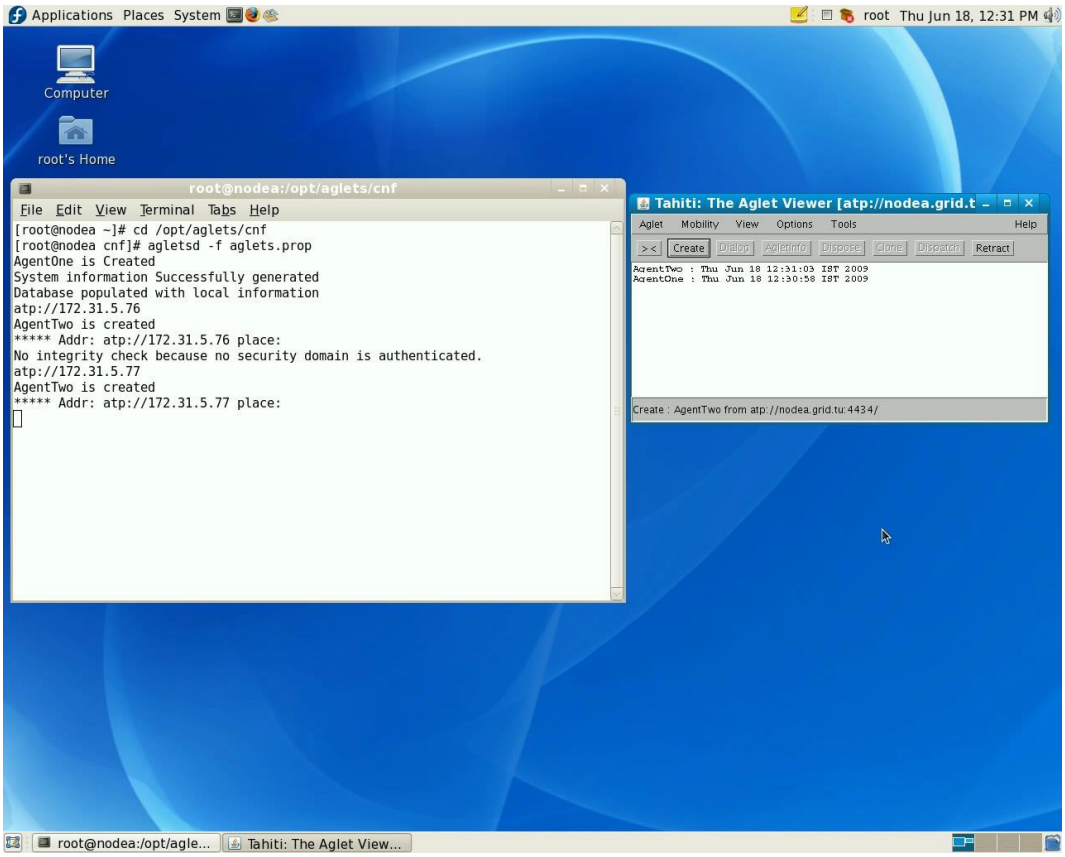

Figure 5.3 AgentOne and AgentTwo in Server Node

Stage Two

AgentTwo arrives on monitored nodes. Figure 5.4 shows AgentTwo's arrival on a monitored node. AgentTwo calls the shell script files to generate system information. This information is stored in a text file. The text file is created in the /opt/aglets/cnf/Monitor folder.

AgentTwo creates AgentThree (Figure 5.5). AgentThree reads the text file and initializes its variables accordingly. Aglets supports weak mobility, aglets runs from its starting point after migration. But aglets variable don't get reinitialized after dispatching. So these data remains intact after AgentThree is dispatched to server node by AgentTwo. If sending of AgentThree to server node fails, AgentTwo disposes the AgentThree. At last AgentTwo disposes itself.

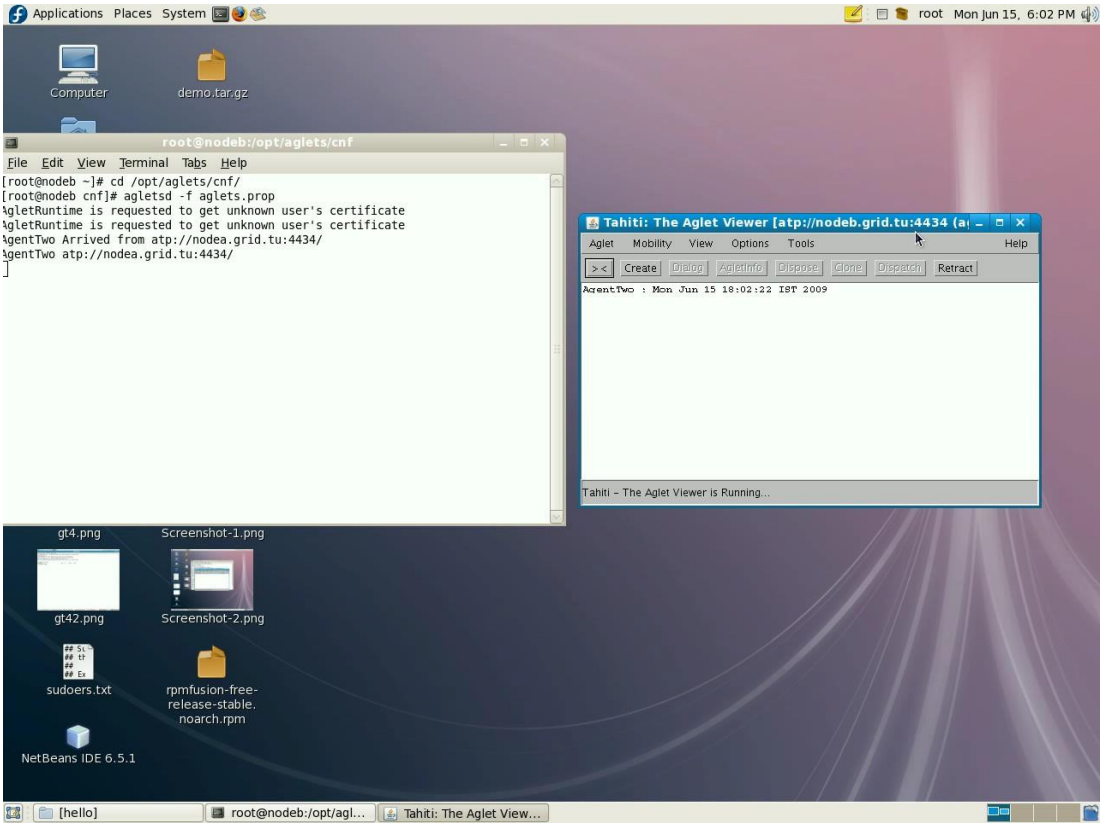

Figure 5.4 AgentTwo on Monitored Node

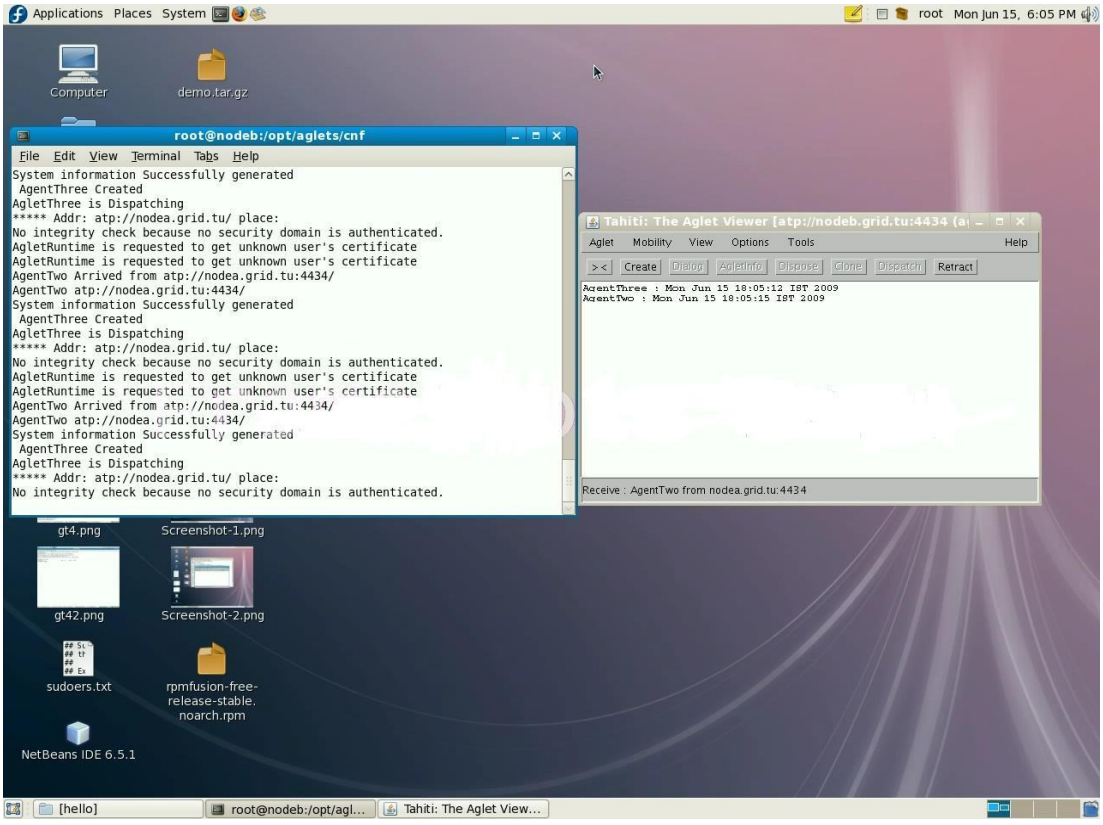

Figure 5.5 AgentTwo and AgentThree on Monitored Node

Stage Three

AgentThree arrives in server node. On server node AgentThree connects with database and updates the database. AgentThree also checks the values to find out memory overloading or CPU overloading. If any overloading found, it also updates overloading data to the database.

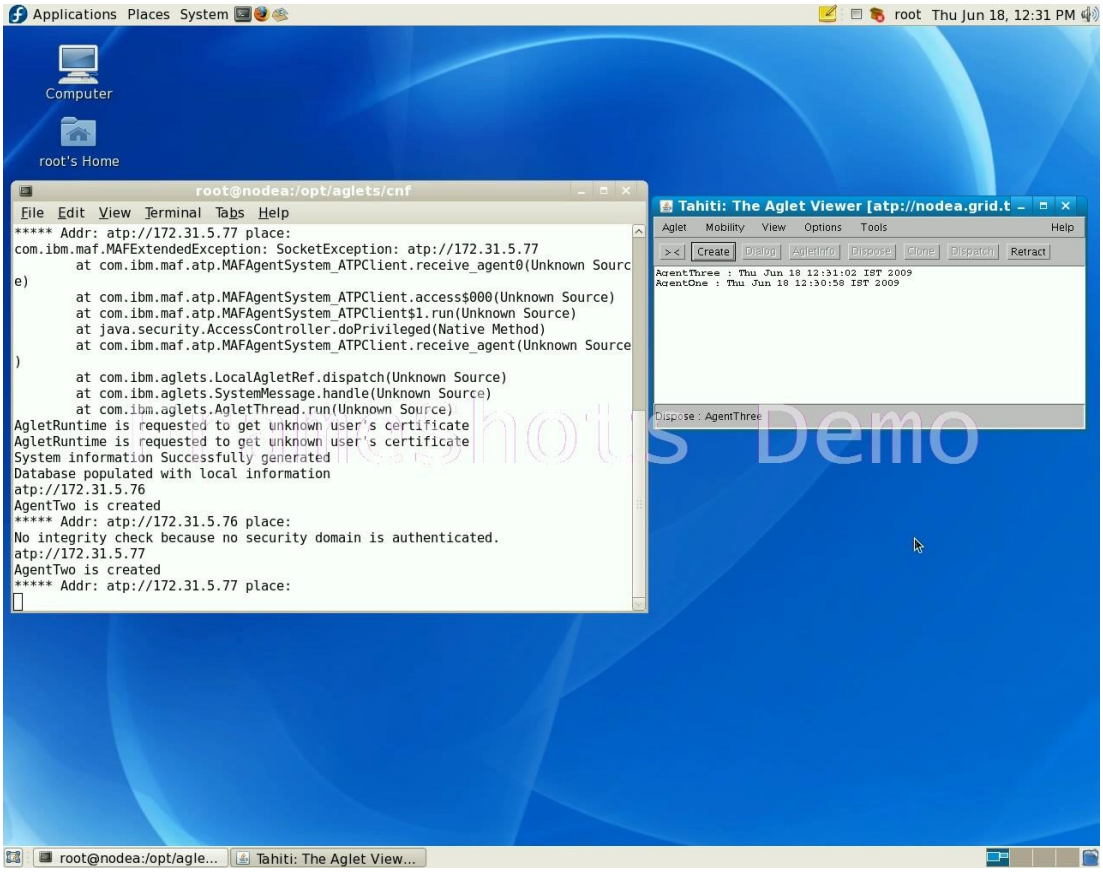

Figure 5.6 AgentOne and AgentThree on Server Node

At last AgentThree disposes itself. The sending of agents, gathering system information and updating the information to database completes with this stages. These three stages repeatedly occur to captures current system information of nodes (local and remote).

The database is also truncated at regular interval by the AgentOne to remove any garbage value from the database.

5.3 Deployment of Web Pages

Web pages used to display system information to the user. These web pages are deployed in the server node. The web pages are hosted on the local network using apache HTTP server.

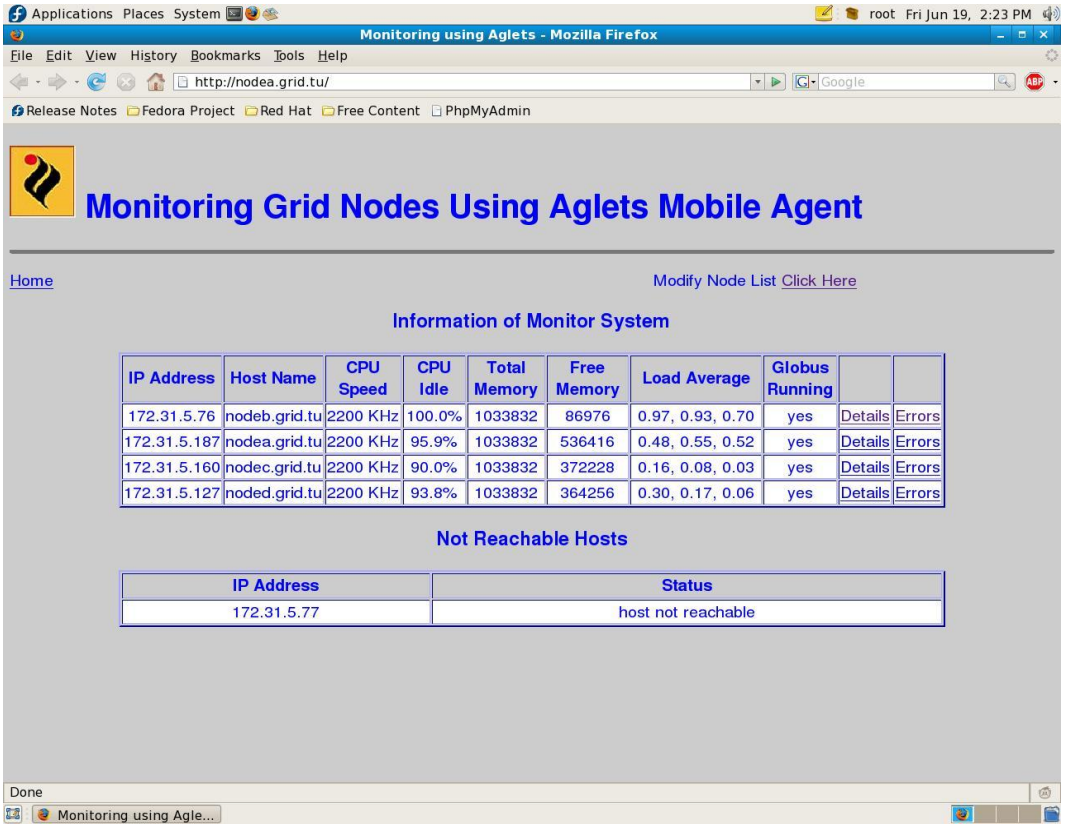

Figure 5.7 The Home Page

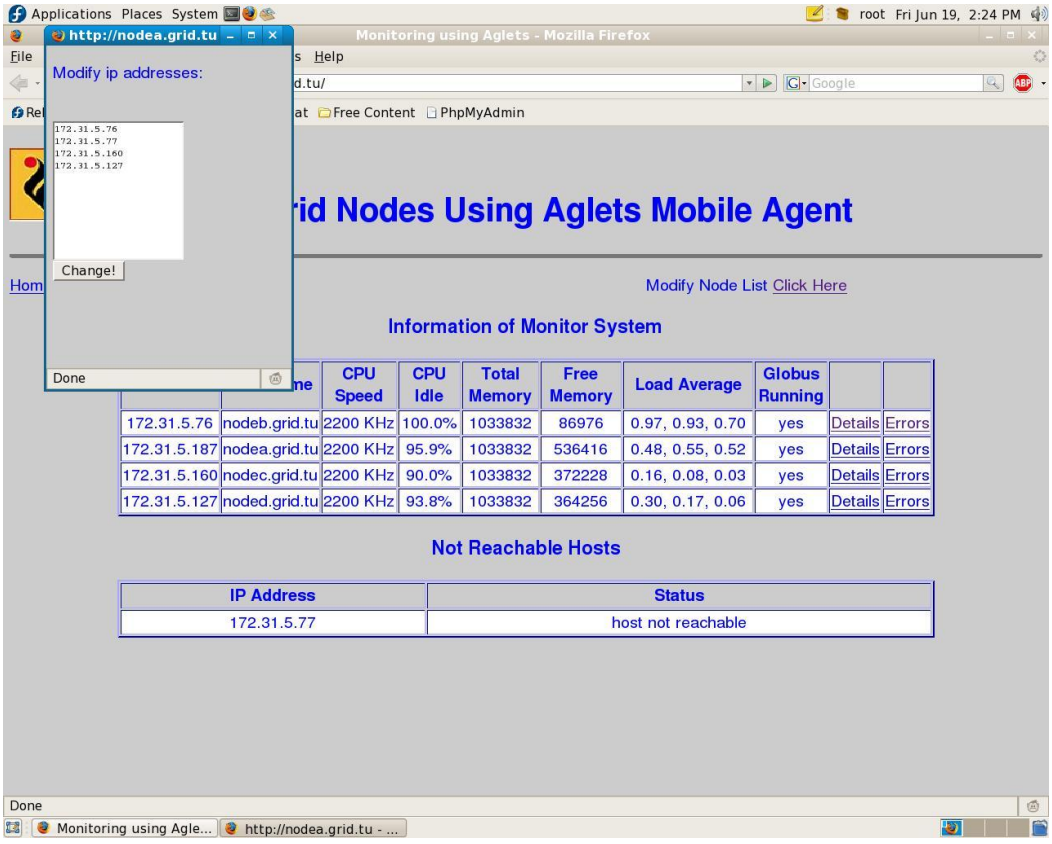

Figure 5.8 Modifying Node List of Monitoring System

Figure 5.7 shows the main page, this page shows list of reachable and unreachable hosts. It also provides general and brief information about the nodes like IP address, hostname, CPU speed, and percentage of idle CPU, total memory, free memory available, load average and status of globus.

Figure 5.7 shows the pop up window to maintain list of monitored nodes. IP address of nodes can be added or removed from the list. Changes in this pop up window results in changes in the ip.txt file of /opt/aglets/cnf/Monitor folder. IP of nodea.grid.tu is not showing in the IP list as nodea.grid.tu is the server node. Server node's information is managed by local information manager. Agents need not to be dispatched to this system, so IP list does not contain the server IP address.

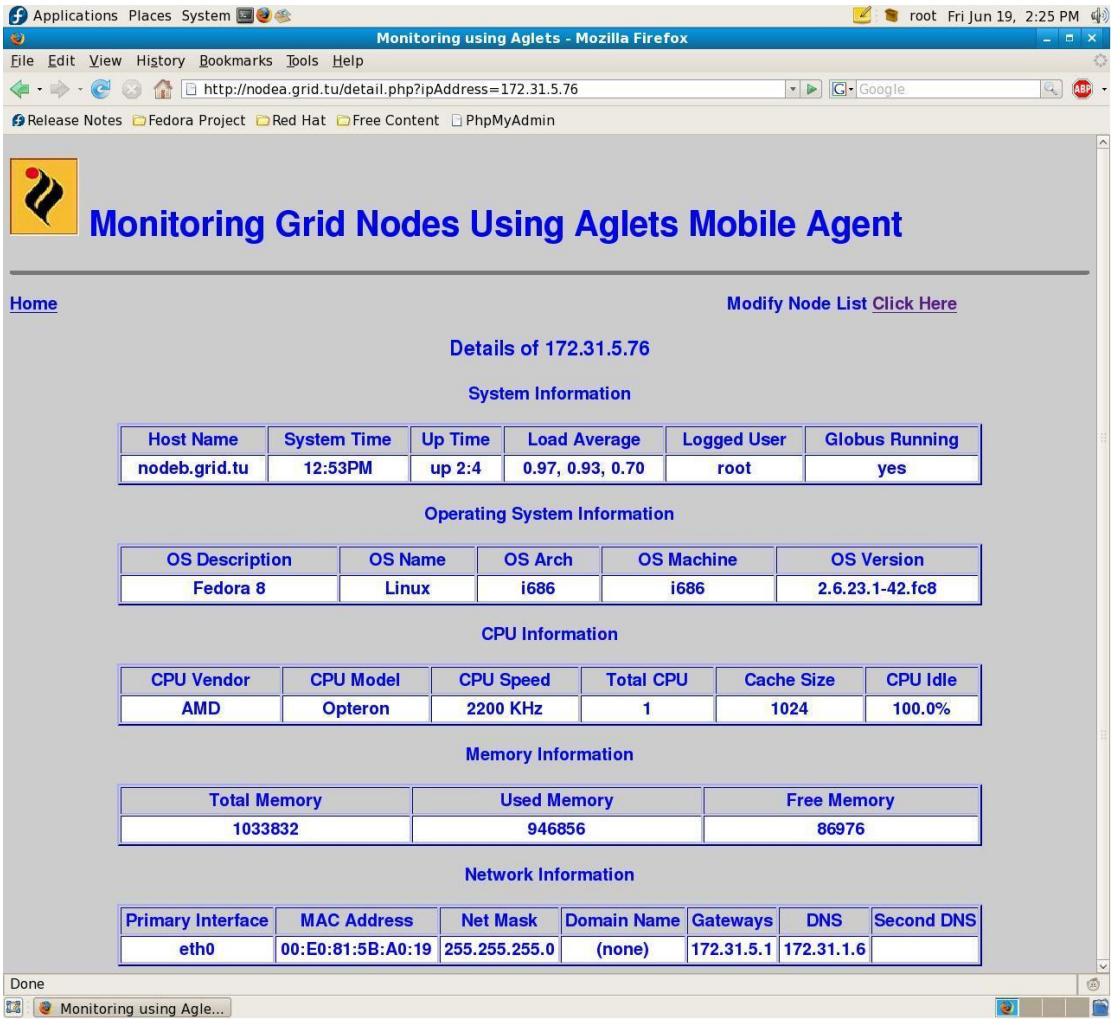

Figure 5.9 Detail Information of Node

Clicking on Details hyperlink will open a page which shows details of a particular node, shown in Figure 5.9. This page shows details of the node having associated IP

address. Details about system, operating system, CPU, memory and network are listed on this page.

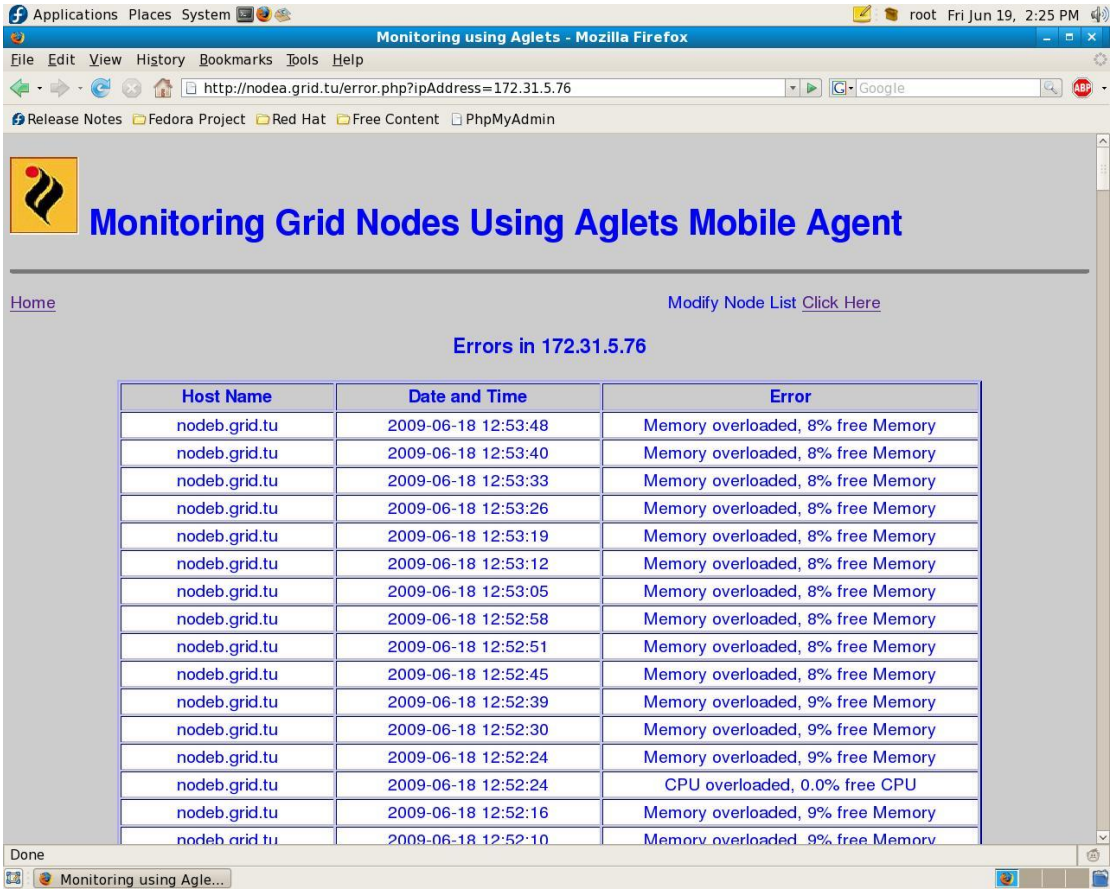

Figure 5.10 Overloading Errors on Node

Figure 5.10 shows the overloading error occurs in a node. It can be opened by clicking on errors hyperlink of main page. It shows errors regarding CPU and memory overloading with time and date.

5.4 Experimental Results

The proposed monitoring system has been tested using the following test cases:

Test Case1 New node is added to the monitoring system.

Test Case2 Node is removed from the node list.

Test Case3 A system is shut down while monitored.

Test Case4 Tahiti server of a node is closed while monitored.

Test Case5 A node is highly overloaded while monitored.

The results of these test cases are given on next page:

Test Result1 The system information of the added system is shown as in Figure 5.7 if the node is up and Tahiti is running. If not then the node is listed under unreachable hosts.

Test Result2 The node is removed from the web pages also. It may take some time as the node's entry from the database is removed by truncating the database.

Test Result3 The node is added to the list of unreachable hosts.

Test Result4 The node is added to the list of unreachable hosts though the node is live and network is also fine. Aglets needs Tahiti server to communicate. As Tahiti is closed, agents from server cannot communicate with this node. As a result this node is declared as unreachable.

Test Result5 As the node is highly overloaded, the value of free memory and free CPU become low. They gone below the threshold value predefined in the AgentThree. As AgentThree finds overloading, table regarding to the error of database is updated. This overloading information can be accessed using the web pages as shown Figure 5.10.

The test results show that the proposed monitoring system achieves all the objectives discussed in the third chapter. The monitoring system is scalable. Any new node can be easily introduced to the monitoring system by adding its IP address in the Node List. Any node can also be removed anytime by removing its entry from Node List. The system is also flexible. Any node can become the server node. To do so it only needs to run the AgentOne using Tahiti.

Conclusions

This chapter discusses about the working of the monitoring system. It shows how aglets work between server node and remote nodes. It also describes how various web pages display the monitoring data and describes the methodology of adding and removing nodes from the node list. At last it presents the test cases, using which the monitoring system has been tested and provides the test result. The test results show the successful design and deployment of the proposed monitoring system. The next chapter concludes this thesis and discusses about the future work.

Chapter 6 Conclusions and Future Work

Monitoring of grid environment is necessary for successful execution of jobs. Grid is very much prone to failures. Execution of a job may become very expensive if the jobs repeatedly fail. To address this problem, a mobile agent based monitoring system for grid systems is proposed in this thesis work. This chapter summarizes the thesis and discusses the future scope of this work.

6.1 Conclusions

Grid computing is not centrally controlled. Any resource or system can join or leave the grid environment at any time. Besides this, grid is also very much prone to failures as it supports heterogeneous resources and protocols. Because of these factoring monitoring is an essential part of grid computing.

This thesis provides discussion on grid monitoring architecture (GMA) and its taxonomy. Along with this, various existing monitoring systems like Ganglia, Hawkeye, Mercury, NWS, RGMA and Globus MDS etc are discussed and a comparison between these monitoring systems on basis of GMA mapping, implementation and security has been reported.

Mobile agents are autonomous, reactive, robust and fault-tolerant. Mobile agents can reduce network load, adapt dynamically to changing environment and they are naturally heterogeneous. Therefore mobile agents are good choice to create grid specific applications.

In this thesis, a mobile agent based monitoring system is proposed for grid environment. Aglets mobile agent platform has been used to create mobile agents. The proposed monitoring system uses Hyperic SIGAR API to generate resource information on grid nodes. The proposed system uses MySQL as backend and web pages created using PHP as frontend.

The proposed mobile agent based monitoring system has been successfully deployed and tested on a globus based grid test bed created in Research Lab of Computer Science and Engineering Department, Thapar University. The experimental results show that the nodes can be easily introduced or removed from the monitoring system; the monitoring system promptly recognizes addition of new nodes. Any node can become the server by hosting AgentOne and no special modification is required.

6.2 Future Work

The monitoring system needs to be evolved with the new releases of future versions of Aglets mobile agent framework and Java. New releases of Aglets may provide a consistent behavior with all versions of JVM then the agents can be modified so that backward compatibility can be achieved. In future, the existing work can be enhanced as suggested:

- The proposed monitoring system has been deployed on the Linux platform and tested using Fedora Core 8. The system can be tested under other Linux distributions also.
- Windows version of the monitoring system can be deployed using the same technologies. As windows does not support shell scripts, the source code of the Hyperic SIGAR need to be modified.
- The monitoring system is meant to use in grid systems. So more grid related information (like grid jobs etc) can be collected and presented.
- The CPU load and memory usage can be depicted using graphs (like ganglia). This could be helpful to the users to understand node load and usage condition.

References

[1] I. Foster, C. Kesselman, et. al, "The Grid: Blueprint for a Future Computing Infrastructure," Morgan Kauffmann Publishers,Inc, San Francisco, California, Second Edition, 2003.

[2] J. D. Sloan, “High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI”, O'Reilly, November 2004

[3] World-Wide Grid (WWG), http://www.gridbus.org/wwg/

[4] A. Sulistio, C. Shin Yeo, R. Buyya, “Visual Modeler for Grid Modeling and Simulation (GridSim) Toolkit4”, Proc. of the 3rd International Conference on Computational Science (ICCS 2003), Springer Verlag Publications (LNCS Series), Melbourne, Australia, June 2 - 4, 2003.

[5] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, Vol 15(3), pp115-128, 2001.

[6] R. Buyya and S. Venugopal, “A Gentle Introduction to Grid Computing and Technologies”, CSI Communications, Vol.29, No.1, Computer Society of India (CSI), Mumbai, India, pp 9-19, July 2005.

[7] G. V. Laszewski, “Grid Computing Enabling a Vision for Collaborative Research”, Proceedings of the 6th International Conference on Applied Parallel Computing Advanced Scientific Computing, Lecture Notes In Computer Science; Vol 2367, pp 37 - 52 , 2002

[8] D. D. Roure, M. A. Baker, N. R. Jennings, N. R. Shadbolt, “The Evolution of the Grid”, John Wiley and Sons Ltd, pp 65-100, 2003.

[9] F. Berman, G. Fox, T. Hey, “The Grid past, present, future”, Grid Computing - Making the Global Infrastructure a Reality, pp 9-50, Wiley and Sons.

[10] I. Foster, J. Geisler, W. Nickless, W. Smith, S. Tuecke, “Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment”, Proceedings of 5th IEEE Symposium on High Performance Distributed Computing. pp 562-571, 1997.

[11] I. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems”, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2006.

[12] Grimshaw A., Wulf W. et al., “The Legion Vision of a Worldwide Virtual Computer”. Communications of the ACM, Vol 40(1), January 1997.

[13] Object Management Group (OMG), http://www.omg.org/

[14] The Common Object Requesting Broker Architecture (CORBA), http://www.corba.org/

[15] Jini, http://www.jini.org

[16] R. Buyya, D. Abramson, J. Giddy, “Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid”, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, USA, 2000.

[17] Towards Open Grid Services Architecture, http://www.globus.org/ogsa/

[18] Types of Grid http://www.gridipedia.eu/types-of-grids.html

[19] J. Unger, M. Haynos, "A visual tour of Open Grid Services Architecture," http://www.ibm.com/developerworks/cn/grid/gr-visual/, 2003. Now available at: http://www.any2any.us/RESOURCES/IBM-OGSA.gr-visual.pdf

[20] Global Grid Forum (GGF), http://www.ggf.org

[21] F. Maciel, J. Treadwell, L. Srinivasan, A. Westerinen, E. Stokes, H. Kreger, D. Snelling, “ Resource Management in OGSA”, March 1, 2005, www.ogf.org/documents/GFD.45.pdf

[22] K. Krauter, R. Buyya, M. Maheswaran, “A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing”, Software: Practice and Experience (SPE), Vol 32(2), Wiley Press, USA, pp 135-164, February 2002.

[23] A. Ali, A. Anjum, A. Mehmood, R. McClatchey, I. Willers, J. Bunn, H. Newman, M. Thomas, C. Steenberg, “A Taxonomy and Survey of Grid Resource Planning and Reservation Systems for Grid Enabled Analysis Environment”, Proceedings of the 2004 International Symposium on Distributed Computing and Applications to Business Engineering and Science, Distributed, Parallel, and Cluster Computing, 2003.

[24] S. Zanikolas, R. Sakellariou, “A Taxonomy of Grid Monitoring Systems”, Future Generation Computer Systems, Vol 21(1), pp 163-188, January 2005.

[23] R. F. Lopes, F. J. da Silva e Silva, “Fault Tolerance in a Mobile Agent based Computational Grid”, Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, CCGRID, 2006.

[26] Jim Gray, “Why Do Computers Stop and What Can Be Done About it”, Technical Report 85.7, Tandem Computers, June 1985. http://www.hpl.hp.com/techreports/tandem/TR-85.7.html

[27] P. Townend, J. Xu, “Fault Tolerance within a Grid Environment,” In Proceedings of AHM2003, page 272, 2003.

[28] C. Inacio, “Software Fault Tolerance”, 18-849b Dependable Embedded Systems, Carnegie Mellon University, Spring 1998. http://www.ece.cmu.edu/~koopman/des_s99/sw_fault_tolerance/

[29] R. Medeiros, W. Cirne, F. Brasileiro, J. Sauvé, “Faults in Grids: Why are they so bad and What can be done about it?,” International Conference on Grid Computing, Proceedings of the 4th International Workshop on Grid Computing, pp 18, 2003.

[30] M. Mansouri-Samani, M. Sloman, “Monitoring Distributed Systems”, IEEE network 7 (6), pp 20-30, 1993.

[31] M. Mansouri-Samani, “Monitoring Distributed Systems (A Survey)”, Tech. Rep. DOC92/93.

[32] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, “A Grid Monitoring Architecture”, GWDPerf- 16–3, Global Grid Forum, August 2002.

[33] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, C. Vistoli, “GridICE: A Monitoring Service for Grid”, Proceedings of the Third Cracow Grid Workshop, Cracow, Poland, pp 220–226, October 27–29, 2003.

[34] R.L. Ribler, J.S. Vetter, H. Simitci, D.A. Reed, “Autopilot: Adaptive Control of Distributed Applications”, Proceedings of the Seventh IEEE Symposium on High-Performance Distributed Computing, pp. 172–179, 1998.

[35] A. Hawkeye,”Monitoring and Management Tool for Distributed Systems”, http://www.cs.wisc.edu/condor/hawkeye/.

[36] Z. Balaton, P. Kacsuk, N. Podhorszki, “Application Monitoring in the Grid with GRM and PROVE”, Proceedings of the International Conference on

Computational Science (ICCS2001), Part I, Vol 2073 of Lecture Notes in Computer Science, Springer-Verlag, San Francisco, CA, USA, p 253, 2001.

[37] B. Tierney, D. Gunter, "NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging", G.S. Goldszmidt, J. Schonwalder (Eds.), Proceedings of the IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), Vol 246 of IFIP Conference Proceedings, Kluwer, pp 97–100, 2003.

[38] R.Wolski, N. Spring, J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", J. Future Generation Comput. Syst. 15 (5/6), pp 757–768, 1999.

[39] M.L. Massie, B.N. Chun, D.E. Culler, "Ganglia Distributed Monitoring System: Design, Implementation, and Experience", Parallel Computing 30, pp 817–840, 2004.

[40] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", Proceedings of 6th IEEE Symposium on High-Performance Distributed Computing, pp 365-375, 1997.

[41] A. Cooke, A.J.G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A.Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher,N. Podhorszki, B. Coghlan, S. Kenny, D. O'Callaghan, "R-GMA: An Information Integration System for Grid Monitoring", Proceedings of the 10th International Conference on Cooperative Information Systems, 2003.

[42] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, M. Thompson, "A Monitoring Sensor Management System for Grid Environments," Cluster Computing, Vol 4(1), pp 19-28, March 2001.

[43] A. Puliafito, O. Tomarchio, L. Vita, "MAP: Design and Implementation of a Mobile Agent Platform,". Journal of System Architecture, Vol 46(2), pp 145–162, 2000.

[44] D. B. Lange, "Mobile Objects and Mobile Agents: The Future of Distributed Computing?" Lecture Notes in Computer Science, ECOOP'98-Object-Oriented Programming, 1998.

[45] D. B. Lange, M. Oshima, “Mobile Agents with Java: The Aglet API”, World Wide Web, Vol 1(3), pp 111 – 121, 1998

[46] D. B. Lange, M. Oshima, “Programming and Deploying Java Mobile Agents with Aglets,” Addison-Wesley, 1998

[47] Apache HTTP Server, http://httpd.apache.org/

[48] Hyperic SIGAR, http://www.hyperic.com/products/sigar.html

[49] Aglets, http://aglets.sourceforge.net/

[50] MySQL, http://www.mysql.com/

[51] The Aglets 2.0.2 User’s Manual, Aglets Development Group, March 12 2009. http://sourceforge.net/projects/aglets/files/User's%20Manual/manual031209.pdf

[52] Java, http://java.sun.com/

[53] PHP, http://www.php.net

Papers Accepted

1. A. Choudhury, I. Chana, "Mobile Agent Based Fault Tolerant System for Grid Environment", to be published in the proceedings of The 2009 International Conference on Grid Computing and Applications hosted by WORLDCOMP 2009, USA, July 13-16, 2009.