A Survey on Grid Technologies and Resource Management Systems

This chapter presents an overview of grid computing, basic grid technologies and current grid activities. It discusses about resource management and some of the available resource management frameworks. It gives a comparative study of GMIS framework with other resource management frameworks.

2.1 Overview of Grid Computing

The term grid was coined to define the idea of exploiting the considerable power represented by the number of computers on the planet and the existence of networks to interconnect them, in order to carry out very long computations for workgroups who need brute force to complete their work, such as auto and airplane designers, scientific researchers, or drug-research firms. To describe it, picture the electricity supply via the electrical grid. Similarly, if we need lots of computing power, we should be able to tap into those resources without worrying about how those resources are generated. Simply plug into the grid and compute.

Grid computing enables the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. Just as an Internet user views a unified instance of content via the Web, a grid user essentially see a single, large virtual computer.

Grid computing is based on an open set of standards and protocols e.g., Open Grid Services Architecture (OGSA) that enables communication across heterogeneous, geographically dispersed environments. With grid computing, organizations can optimize computing and data resources, pool them for large capacity workloads, share them across networks and enable collaboration.
Grid computing appears to be a promising trend for three reasons:

(i) Its ability to make more cost-effective use of a given amount of computer resources
(ii) As a way to solve problems that can't be approached without an enormous amount of computing power
(iii) It suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as collaboration towards a common objective.

Benefits of grid computing:

- Infrastructure optimization
- Increase access to data and collaboration
- Highly available infrastructure
- Accelerate time to results
- Enable collaboration and promote operational flexibility

To further illustrate this environment and often times very complex set of technology challenges, let us consider a scenario, which will begin to examine the many values of a grid computing solution environment.

A group of scientists studying the atmospheric ozone layer collect huge amounts of experimental data, each day. These scientists need efficient and complex data storage capabilities across wide and geographically dispersed storage facilities, and they need to access this data in an efficient manner based on the processing needs. This ultimately results in a more effective means of performing important scientific research. Grid offers a very lucrative solution for this type of problems.

What is not a grid? A cluster, a network attached storage device, a scientific instrument, a network; these are not grids. Each might be an important component of a grid, but by itself, it doesn't constitute a grid.
2.1.1 Ancestors of the Grid

Although the grid can trace its roots back to the operating system Multics [Multics], the immediate ancestor of the Grid is “meta computing” [Meta computing], a term that dates back to around 1990, and was used to describe projects aimed at interconnecting supercomputer centers in order to combine the processing power of multiple supercomputers. Two cutting edge Meta computing projects both conceived in 1995 were called Factoring via Network-Enabled Recursion (FAFNER) [FAFNER] and Information Wide Area Year (I-WAY) [IWAY, 1996]. Each in its own way influenced the evolution of some of the key grid technology projects ongoing today.

FAFNER was a project, which aimed at factorizing very large numbers, a computationally intensive challenge which is very relevant to digital security. Since this is a challenge which can be broken into small parts, even fairly modest computers can contribute useful power. Many of the techniques developed for splitting up and distributing a big computational problem were forerunners of the technology used for “cycle scavenging” software.

I-WAY was a project that aimed at linking supercomputers, but using only existing networks, not building new ones. One of the innovations of the I-WAY project was developing a computational resource broker, conceptually very similar to the resource brokers being developed for the grid today. I-WAY strongly influenced the development of the Globus project, which is at the core of most grid activities, as well as the LEGION [Grimshaw, 1997] project, an alternative approach to distributed super computing.

The grid itself was born at a workshop at Argonne National Laboratory in 1997, called “Building a Computational Grid”, which was followed by the publication of “The Grid: Blueprint for a New Computing Infrastructure” by Ian Foster of Argonne National Laboratory and Carl Kesselman of the University of Southern California [Ian Foster, Carl Kesselman, 1999]. Ian Foster is often known as the father of grid computing.
2.1.2 Architecture of Grid Computing

The architecture for grid computing is defined in the Open Grid Services Architecture (OGSA), developed through the Global Grid Forum (GGF) [GGF]. OGSA defines what grid services are and the overall structure and services to be provided in grid environments.

The Open Grid Services Infrastructure (OGSI) is a formal specification of the concepts described by the OGSA. More than two dozen working groups at the Global Grid Forum are busy defining an array of grid standards in areas like applications and programming models, architecture, data management, security, performance, and scheduling and resource managements.

The architecture of the grid is described in terms of layers, each providing a specific function. In general, the higher layers are focused on the user (user-centric) whereas the lower layers are more focused on computers and networks (hardware-centric). The OGSA grid architecture is shown in the following figure 2.1.

Figure 2.1: OGSA Grid Architecture [Foster, I, C. Kesselman, 2002]
As shown in above figure 2.1, at the base of everything, the bottom layer is the network layer, which assures the connectivity for the resources in the grid. On top of it lies the resource layer, made up of the actual resources that are part of the grid, such as computers, storage systems, electronic data catalogues, which can be connected directly to the network.

The middleware layer provides the tools that enable the various elements (servers, storage, networks, etc.) to participate in a unified grid environment. The middleware layer can be thought of as the intelligence that brings the various elements together.

The highest layer of the structure is the application layer, which includes all different user applications (science, engineering, business, financial), portals and development toolkits supporting the applications.

In most common grid architectures, the application layer also provides the serviceware, the sort of general management functions such as measuring the amount a particular user employs the grid, billing for this use, and keeping accounts of who is providing resources and who is using them - an important activity when sharing the resources of a variety of institutions amongst large numbers of different users. The serviceware is in the top layer, because it is something the user actually interacts with, whereas the middleware is a "hidden" layer which is transparent to the user. In order to achieve the benefits of grid computing the applications architecture need to consider grid related services and protocols available. If traditional applications that run on a single PC or server have to run in grid environment then they should be adapted to grid environments which require users to invest some effort. However once traditional applications are adapted to grid environment thousands of people will be able to use the same application and run in trouble free on this grid.

2.2 Basic Grid Technologies

The grid is inherently very complicated. Factors that contribute to this complication include many network types, incompatible hardware architectures, different operating system security mechanisms and different protocol support in many
programming languages. Hence, the solution to create a grid is understandably complicated and is a nontrivial task. Many efforts have been made to construct a homogeneous view of this heterogeneous environment. Some of such popular efforts are discussed in this section.

2.2.1 Globus Toolkit

Globus toolkit [Foster, I, C. Kesselman, 2001] [Foster, I, C. Kesselman, 2002] is a free and open source grid toolkit. It implements the Open Grid Services Infrastructure (OGSI) standards and provides tools to build, develop, deploy, and manage grid services. The open source Globus Toolkit is a fundamental enabling technology for the "Grid," letting people share computing power, databases, and other tools securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over whom and when these resources can be used.

Grid tools that are available in the toolkit can be broadly classified into the following:

- **Infrastructure**: Infrastructure components include file systems, schedulers and resource managers, messaging systems, security applications, certificate authorities, and file transfer mechanisms like GridFTP.
• **Directory Services:** Systems on a grid must be capable of discovering what services are available to them. This means that the grid systems must be able to define and monitor a grid's topology in order to share and collaborate. Many grid directory services implementations are based on past successful models, such as Lightweight Directory Access Protocol (LDAP) [OpenLdap], Domain Name Service (DNS), network management protocols, and indexing services.

• **Schedulers and load balancers:** One of the main benefits of a grid is its ability to maximize efficiency. Schedulers and load balancers provide this function and more. Schedulers ensure that jobs are completed in some order (for instance priority, deadline, priority) and load balancers distribute tasks and data management across systems to decrease the chance of bottlenecks.

• **Security:** Security in a grid environment provides authentication and authorization, controlling who can access a grid's resources. For instance, message integrity and message confidentiality are crucial to financial and healthcare environments.

The Globus toolkit has grown through an open-source strategy similar to the Linux operating systems, and distinct from proprietary attempts at resource-sharing software. This encourages broader, more rapid adoption and leads to greater technical innovation, as the open-source community provides continual enhancements to the product.

The main drawback of Globus toolkit is that it is a very complicated toolkit. While it is a feature rich, it can be very time consuming to configure, install and operate. Usually it takes several hours (approximately 7 hours) to install. It is unstable on some platforms and provides only a base support e.g., Sun Solaris. Globus toolkit is known to have better performance on SuSE or RedHat Linux operating systems.

### 2.2.2 IBM Grid Tool box

IBM Grid Toolbox, [IBM Redbook, 2004] developed by IBM, is a free grid Integration Development Environment (IDE) and implements OGSI standards. The major...
The advantage of this toolbox is that its installation process is very simple. It ships both wizard-based and silent installation methods. Users are relieved from the pain to install prerequisite packages like in other open source software. The IBM Grid Toolbox installation on Pentium-based Personal Computer (PC) or xSeries takes around 40 minutes. The tool was built entirely based on the OGSI specification.

The toolbox is provided with enhancements and tools which assist with building, packaging and deploying grid services and applications. This feature greatly reduces the complexity of grid services development and deployment. It consists of the following:

- A hosting environment capable of running grid services and collaborating with other grid participants in running large tasks.
- A set of tools to manage, monitor, and administer grid services and the grid hosting environment, including a web-based interface called IBM Grid Services Manager.
- A set of APIs and development tools to create and deploy new grid services and grid applications.
- A set of tools to simplify the installation process and the integration of the embedded middleware, such as IBM WebSphere Application Server.

The IBM Grid Toolbox is a collection of components. The following overview of components suggests the scope of features that the IBM Grid Toolbox brings to grid developers and administrators:

- **Single installation process:** The IBM Grid Toolbox ships both wizard-based and silent installation methods, which provide simple one-off or bulk installation around a network.

- **Grid services runtime based on the OGSI specification:** An embedded version of the IBM WebSphere Application Server – Express V5.0.2 is provided as the grid services container. It replaces the stand-alone container that is provided by Globus toolkit for commercial-grade support.
- **Management interface:** A browser-based interface called the IBM Grid Services Manager provides easy grid-wide management for administrators.

- **Security:** An enhanced certificate-based grid security infrastructure.

- **Configuration and administration commands:** Command-line based scripts for common actions are provided for administrators.

- **Development tools:** Enhancements and tools are provided that assist with building, packaging, and deploying grid services and applications.

- **Additional and enhanced grid services:** IBM provides additional functionality including discovery via service group, policy management, and Common Management Models (CMM) Services.

### 2.2.3 SUN Grid Engine

SUN’s Grid solution, SUN Grid Engine (SGE) [SUN Grid] is an open source but not free grid IDE and does not implement OGSI standards. SGE contains four types of hosts: Master hosts, Execution hosts, Administration hosts, and Submit hosts. The Master host runs master and scheduler daemons, which control all SGE components, such as queues and jobs. Execution hosts have permission to execute SGE jobs. Administration hosts carry out administrative activity for the grid system. Submit hosts allow for submitting and controlling batch jobs only. For example, a logged in user can submit jobs via "qsub" and control job status via "qstat". Besides the command-line interface, SGE system also provides graphical user interface, called "QMON".

SGE allows users to submit jobs to queues. Master and scheduler daemons determine which execution host requires to run the tasks. In case some execution hosts crash, SGE has the facility of using other execution hosts to fail over transparently, where as the other open source tools don't have this facility. In order to achieve reliability and scalability for a grid, a SGE could be used as a grid scheduler.

SGE provides the following features [SGE Features]:
- **Multi-clustering with Service Domain Manager:** The SGE Service Domain Manager allows two or more SGE clusters to share resources. It offers optimal service level to users. It delivers built-in elasticity by adding or removing hosts from each local cluster according to service level objectives.

- **Advance Reservation:** In SGE users can request and reserve grid resources in advance, such as hosts, memory, or licenses, for a specified time window.

- **Scalability:** SGE scales up to 63,000 core CPUs.

- **Improved Interactive Job Support:** SGE enables better accounting and monitoring of the interactive users in the grid. It provides a faster and more robust connection.

- **Support massively parallel jobs:** SGE supports robust parallel jobs spanning tens of thousands of CPU cores. It is faster in start-up of large parallel jobs.

- **Array Task Dependency:** SGE allows for faster production times. It enables special effects and rendering to be done faster than with competitive software.

- **Accounting and Reporting Console:** SGE provides a clear report of the usage of all SGE clusters throughout the entire organization.

- **Better Integration with Sun Software:** SGE provides better manageability and quality of service when the SGE software runs on the Solaris 10 operating system.

SGE is a system for cluster management, rather than real grid resource management. It does not implement OGSI standards, nor support grid services. Unlike IBM Grid Toolbox, SGE could not allow grid or web services deployment, but only support script-based job management.
2.2.4 Avaki

Avaki [IBM Redbook, 2003] is a commercial grid IDE developed by Avaki Inc. Ltd. Avaki was formed to commercialize the technology derived from the Legion project [Grimshaw, 1997]. It implements the OGSI standards and provides the tools to build, develop, deploy, and manage grid services. The AVAKI grid solution pools heterogeneous resources across an enterprise, making them easily accessible on-demand. With AVAKI grids, research and development organizations can overcome problems such as, the need for secure access to fresh and consistent data; a uniform environment that is easy for researchers to use and for IT professionals to administer and a highly available computing infrastructure that delivers maximum application performance and reliability.

At a high level, Avaki’s vision is one of a ubiquitous, pervasive computing infrastructure, one that enables secure and transparent access to all computing resources. To that end, Avaki delivers grid software solutions, also known as middleware, that bring together data, compute, and application resources from multiple locations and administrative domains for industries such as life sciences, manufacturing, and oil & gas. They have products for both data grids and compute grids, but are focusing a majority of their technology on data grids (80 percent of the projects they’re working on these days are data grid related projects). In addition, although they solve problems across verticals, they are focused primarily on domestic pharmaceutical companies.

Avaki’s products include Avaki Data Grid, Avaki Compute Grid, and Avaki Comprehensive Grid.

**Avaki Data Grid** provides wide-area access to distributed data, making it easily available for users, applications, and business process. Data grid also manages security and usage policies such as enabling multi-company data access and collaboration which are key to the forward-looking vision of virtual organizations. A data grid, with its distributed nature, also enhances remote processing by efficiently bringing the data closer to the application.
Avaki Compute Grid aggregates processing resources within or across locations, leveraging existing infrastructure. Authentication, security, and usage policies for compute grid are similar to those for data grid. Job scheduling and queuing are also important characteristics of compute grid, aiding in the processing of multiple jobs over a common and shared processing infrastructure.

Avaki Comprehensive Grid combines features of compute grid and data grid to provide wide-area access to processing resources, data, and applications.

The components of the Avaki solution include:

- **Grid Servers** contain house authorization information. Each grid domain has a grid server called a Grid Domain Controller
- **Share Servers** authorize sharing of data on the grid
- **Data Grid Access Server** for High-performance caching
- **Proxy Server** is for Firewall proxy
- **Client Components** like Web User Interface, command-line client

Much of the underlying protocol and messaging is done via Hyper Text Transfer Protocol (HTTP) and Secure Socket Layer (SSL). Remote Procedure Calls provide credentials to the access control lists.

One of the keys to Avaki’s technology is its intelligent caching approach. This is key in keeping the authoritative copy of the data with the owner, while still allowing sharing and access with remote users for very large data sets. A simplistic view of Avaki is as, basically, a distributed database solution. Avaki is soon to announce a deal with a top 10 pharmaceutical that will be one of the largest data grid deployments to date (over 500 collaborations with multi-tier partnership relationships involved with outsourcing pharmaceutical processes). Due to the data-intensive nature of Avaki’s solution, their primary opportunities are with enterprise grids and wide area grids.
2.2.5 Microsoft .NET

The Microsoft .NET platform provides a useful way to construct grid systems that make use of the emerging interest in web services. Various tools are available for reducing the complexity of implementing web services, and providing support for messaging between such services. For instance, Visual Studio .NET provides a development environment for writing such services in C# and J# (Microsoft’s version of Java). The importance of such tools is essential to enable a wider adoption and usage of web services-oriented grid software.

The .NET platforms also support a number of additional features that could be useful to construct grid systems, particularly for enabling a number of different platforms to interact. The “Common Language Runtime” (CLR) in .NET provides an execution environment that can be ported to a number of different machine architectures, and supports the execution of programs written in a variety of different languages. The CLR adopts a similar approach to the Java Virtual Machine, for enabling code developed on one machine to be ported (without re-compilation) to another. The .NET platform also provides a Just-In-Time (JIT) compiler that allows dynamic performance improvements during the execution of a program, and is facilitated in this by the CLR. Microsoft is also aiming to make a version of the CLR open-source, to enable community input into this activity.

2.2.6 Comparison of Various Grid Technologies

The below table 2.1 shows a side-by-side comparison of above discussed grid technologies:

<table>
<thead>
<tr>
<th></th>
<th>Globus</th>
<th>IBM Grid Toolbox</th>
<th>SUN Grid Engine</th>
<th>Avaki</th>
<th>Microsoft .NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Price</td>
<td>Free</td>
<td>Free</td>
<td>Commercial</td>
<td>Commercial</td>
<td>Commercial</td>
</tr>
<tr>
<td>OGSI Standards</td>
<td>Support</td>
<td>Support</td>
<td>No</td>
<td>Support</td>
<td>Support</td>
</tr>
<tr>
<td>Grid services</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Platforms</td>
<td>Linux, All Unix</td>
<td>Linux, AIX</td>
<td>Linux, Solaris</td>
<td>Linux, Windows</td>
<td>Windows, All UNIX</td>
</tr>
</tbody>
</table>

27
### Table 2.1: Comparison of Different Grid Technologies

<table>
<thead>
<tr>
<th>Installation</th>
<th>Not Easy</th>
<th>Easy</th>
<th>Easy</th>
<th>Easy</th>
<th>Easy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Tools</td>
<td>Some Scripts</td>
<td>Enhanced Tooling</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

2.3 Current Grid Activities

There are innumerous grid activities going on in the world. Among them, some concentrate on defining standards, some on development of software tools, and others on services, or different scientific applications. Following is a brief mention of some of the important ones:

- **Grid Standards Projects** – focusing on refining the grid standardization process and defining best practice guidelines for the scientific and industry usage of grid. The most prominent among such organizations is Global Grid Forum (GGF).

- **Grid-Tech Projects** - primarily involved in development of grid-enabling technology, such as middleware and hardware. Some examples are CONDOR, BIOGRID etc.

- **Test Bed Projects** - devoted to develop and maintain a working test beds using existing grid technology like European Union: DataGrid, EUROGRID etc.

- **Grid Fora Projects** - devoted to catalyze, stimulate and foster collaboration on grid related projects. They are aiming to share technology, resources and knowledge in order to promote grid technologies and applications like PRAGMA.

- **Field specific Projects** - projects devoted to explore and harness grid technology in the context of specific fields of scientific research such as BIOINFORMATICS and E-SCIENCE program.

- **Grid Portals** - Internet portals to grid related activities. One such portal is EnterTheGrid which is an extensive dictionary to grid related activities.
• ...@home - Internet computing projects such as Search for Extra Terrestrial Intelligence (SETI) experiment uses Internet distributed computing to search for intelligent life outside Earth.

Some popular grid projects like GGF, CONDOR, European Union DataGrid, PRAGMA, and Grid Bus are briefly discussed in subsequent sections.

2.3.1 Global Grid Forum

Global Grid Forum (GGF) [GGF] was established as a public community forum for the discussion of grid technology issues. The GGF enables a means of coordinating grid computing technology efforts, promoting reuse and interoperability, and sharing the results. As of now, there are more than 400 organizations involved with GGF from around the world. This includes scientific research institutions, universities, and commercial organizations.

The GGF’s primary objective is to promote and support development, deployment, and implementation of grid technologies and applications via creation and documentation of best practices – specifications, use cases, architecture, and implementation guidelines.

The basic goals of the GGF are to:

• Create an open process for the development of grid agreements and specifications
• Create grid specifications, architecture documents, and best practice guidelines
• Manage and version controls the documents and specifications
• Handle intellectual property policies
• Provide a forum for information exchange and collaboration
• Improve collaboration among the people involved with grid research, grid framework builders, grid deployment, and grid users
• Create best practice guidelines from the experience of the technologies associated with grid computing
• Educate on advances in the grid technologies and share experiences among the people of interest

The organization consists of different work areas, with research groups and work groups for each area. The work groups are the main activity centers of the GGF. These work groups are created to address a research, implementation, and operational area related to the infrastructure for building any grid.

The major work areas of the GGF are:

• Application and programming environments
• Architecture
• Data
• Information systems and performance
• Peer-to-peer: desktop grids
• Scheduling and resource management
• Security

One of the major activities in GGF that is attracting the grid community is the architecture model based on the open standard web service architecture, called Open Grid Service Architecture (OGSA). With open standards as the foundation and software integration, OGSA has emerged as the core grid technology for future resource sharing, especially with the newly added commercial dimension to grid solutions.

2.3.2 Condor

Condor [Condor] is a tool for harnessing the capacity of idle workstations for computational tasks. Condor is well suited for parameter studies and high throughput computing, where jobs generally do not need to communicate with each other.

Condor can be classified as a specialized workload management system for computation-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and
resource management. Upon receiving serial or parallel jobs from the user, the Condor system places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.

Condor can manage a cluster of dedicated compute nodes. It is suitable for effectively harnessing the CPU power from idle workstations. Condor has mechanisms for matching resource requests (jobs) with resource offers (machines).

While Condor software tools focus on harnessing the power of opportunistic and dedicated resources, Condor-G is a derivative software system, which leverages the software from Condor and Globus with major focus on the job management services for grid applications. This is a combination of inter-domain resource management protocols of Globus (GRAM, Index services) with the intra-domain resource management methods of Condor.

Condor software is used by both scientific and commercial organizations. The major scientific initiative that uses Condor includes NSF Middleware Initiative (NMI), Grid Physics Network (GriPhyN), International Virtual Data Grid laboratory (iVDGL), TeraGrid, and so on. Some of the prominent commercial uses of Condor software involve solving computational grid computing problems, as done by Micron Technologies, CORE Digital Pictures, and NUG30 Optimization problem solver.

2.3.3 European Union: DataGrid

DataGrid [DataGrid, 2002] is project funded by the European Union that aims to enable access to geographically distributed computing power and storage facilities belonging to different institutions. This will provide the necessary resources to process huge amounts of data coming from scientific experiments in different disciplines.

The three real data-intensive computing applications areas covered by the project are:

- High Energy Physics
- Biology and Medical Image Processing
Earth Observations

High Energy Physics: One of the main challenges for High Energy Physics is to answer long standing questions about the fundamental particles of matter and the forces acting between them. In particular, the goal is to explain why some particles are much heavier than others, and why particles have mass at all. To that end, CERN is building the Large Hadron Collider (LHC), on of the most powerful particle accelerators.

The serial on LHC will generate huge amount of data. The DataGrid project is providing the solution for storing and processing this data. A multi-tiered, hierarchical computing model is adopted to share data and computing power among multiple institutions. The Tier-0 center is located at CERN and linked by high-speed networks to approximately 10 major Tier-1 data processing centers. These will fan out the data to a large number of smaller ones (Tier-2).

Biology and Medical Image Processing: The storage and exploitation of genomes and the huge flux of data coming from post-genomics puts growing pressure on computing and storage resources within existing physical laboratories. Medical images are currently distributed over medical image production sites (radiology departments, hospitals). As of today, since there is no standard for sharing data between sites, there is an increasing need for remote medical data access and processing.

The DataGrid project’s biology test-bed is providing the platform for the development of new algorithms on data mining, databases, code management, and graphical interface tools. It is facilitating the sharing of genomic and medical imaging databases for the benefit of international cooperation and health care.

Earth Observations: The European Space Agency missions download 100 gigabytes of raw images per day from space. Dedicated ground infrastructures have been set up to handle the data produced by instruments onboard the satellites. The analysis of atmospheric ozone data has been selected as a specific test-bed for the DataGrid. Moreover, the project demonstrates an improved way to access and process large volumes of data stored in distributed European-wide archives.
2.3.4 Pacific Rim Application and Grid Middleware Assembly (PRAGMA)

The Pacific Rim Application and Grid Middleware Assembly (PRAGMA) [PRAGMA] was formed to establish sustained collaborations and advance the use of the computational grid among a community of investigators at the leading institutions around the Pacific Rim. Applications are the key focus of PRAGMA, with the intent of using applications to bring together the key infrastructure and middleware necessary to advance the goals of the application. **University of Hyderabad** has become one of the founding partners of such an international collaboration.

PRAGMA accomplishes its mission primarily by,

- conducting joint projects that develop grid middleware to advance applications,
- sharing resources to create a test bed,
- addressing scheduling and allocation issues across institutional and international boundaries.

In addition, PRAGMA is committed to disseminating the results of its efforts to the broader community and to work with regional and international groups to enhance the overall grid infrastructure and to promote global collaboration.

2.3.5 Grid Bus

The Gridbus project [Buyya, VenuGopal, 2004] is engaged in the design and development of grid middleware technologies to support eScience [T.Hey, A.E. Trefethen, 2002] and eBusiness applications. These include visual grid application development tools for rapid creation of distributed applications, competitive economy-based grid scheduler, cooperative economy based cluster scheduler, web-services based Grid Market Directory (GMD), grid accounting services, Gridscape for creation of dynamic and interactive test-bed portals, G-monitor portal for web-based management of grid applications execution, and the widely used GridSim toolkit for performance evaluation. Gridbus project has developed Windows .NET-based desktop clustering software and grid job web services to support the integration of both Windows and Unix-based systems.
class resources for grid computing. A layered architecture for realization of low-level and high-level grid technologies is shown in figure 2.2.

GridBus contain following components:

- **Visual Parametric Modeler**: A graphical environment for application parameterization.

- **G-Monitor**: A web portal to manage execution of applications on grids using remote brokers.

- **Grid Service Broker**: An economy-based data grid broker for scheduling distributed data oriented applications across Windows and Unix-variant grid resources.

- **Grid Market Directory**: A directory for publication of grid service provides and their services.
- **Grid Bank**: A grid accounting, authentication, and payment management infrastructure.

- **Gridscape**: A tool for the creation of interactive and dynamic grid test bed web portals.

- **Alchemi**: A .NET-based desktop grid framework.

- **Libra**: An economy based scheduler for clusters.

- **GridSim**: A toolkit for modeling and simulation of global grids.

## 2.4 Resource Management

A resource on a grid could be any entity that provides access to a service. This could range from compute servers to databases, scientific instruments and applications. In a heterogeneous environment like a grid, resources are generally owned by different people, communities or organizations with varied administrative policies, and capabilities. Resource management simplifies the process of naturally obtaining and managing access to resources to all its users. In the grid computing environment resource management is a collection of software components that let users to access heterogeneous resources transparently, without having to worry about availability, access methods, security issues and other policies.

Consider a job management system, where the resource management feature identifies the job, allocates the suitable resources for the execution of the job, partitions the job if necessary, and provides feedback to the user on job status. This job scheduling process includes moving the data needed for various computations to the appropriate grid computing resources, and mechanisms for dispatching the job results.

It is important to understand multiple service providers can host grid computing resources across many domains, such as security, management, networking services, and application functionalities. Operational and application resources may also be hosted on different hardware and software platforms. In addition to this complexity, grid
computing middleware must provide efficient monitoring of resources to collect the required matrices on utilization, availability, and other required information.

The following brief and partial list provides a resource specific characterization of capabilities.

- **Computational resources management**: Mechanisms are required for starting programs for monitoring and controlling the execution of the resulting processes. Management mechanisms that allow control over the resources allocated to processes are useful, as are advance reservation mechanisms. Enquiry functions are needed for determining hardware and software characteristics as well as relevant state information such as current load and queue state in the case of scheduler-managed resources.

- **Storage resources management**: Mechanisms are required for storing and retrieving of files. Third-party and high-performance (e.g., striped) transfers are required. These mechanisms used for reading and writing subsets of a file and/or executing remote data selection or reduction functions. Management mechanisms that allow control over the resources allocated to data transfers (space, disk bandwidth, network bandwidth, CPU) are required. Enquiry functions are needed for determining hardware and software characteristics as well as relevant load information such as available space and bandwidth utilization.

- **Network resources management**: Management mechanisms that provide control over the resources allocated to network transfers (e.g., prioritization, reservation) can be useful. Enquiry functions should be provided to determine network characteristics and load.

- **Code repositories management**: Code repositories are specialized form of storage resources require mechanisms for managing versioned source and object code: for example, a control system such as Concurrent Versions System (CVS).
• **Catalogs management:** Catalogs are specialized form of storage resources that require mechanisms for implementing catalog query and update operations: for example, a relational database.

Usually the resource management scenarios include resource discovery, monitoring and brokerage.

**2.4.1 Resource Discovery**

For resource sharing and integration in grid computing environment, the abilities of discovering, allocating, negotiating, monitoring and managing network accessible resources are essential to achieve various end to end or global qualities of services. Thus grid resource discovery becomes a basic component of grid resource management, which is a core of grid and provides a transparent global resource view for the resource schedule and application.

In a grid system, the resource discovery service operates in conjunction with the resource management service. Resource discovery is the process of identifying and locating existing resources on grid. It is used to search information about resources available in the grid network. Schedulers or brokerage service use the resources information discovered by the resource discovery service.

**2.4.2 Resource Monitoring**

In order to construct and execute applications on grid resources, an understanding of the type and availability of grid resources is necessary. Grid monitoring services provide information about the status of grid resources. These services are used to describe grid resources. Lack of knowledge about resources will hamper resource scheduling, allocation and usage. Hence it is important that grid information service failures are reported and dealt with in a timely fashion.

Resource monitoring is a vital function in a distributed system, particularly when that system spans multiple locations as in that context no one is likely to have detailed knowledge of all components. Resource monitoring allows us to detect and diagnose the
many problems that can arise in such contexts, for example CPU load of a computing node is more than 80%. This task requires the ability to collect information from multiple, perhaps distributed, information sources.

### 2.4.3 Resource Brokerage

Resource Brokerage is defined as the process by which a node or machine in a distributed system becomes aware of the attributes or capabilities of the other nodes that are part of the system. In grid systems, the resource brokerage process is mainly used by a node to find out the “best” set of candidate resources that can execute a job or provide a specific service. The efficiency of the performance of brokerage process can be improved by observing that a grid is likely to consist of a variety of heterogeneous machines and networks.

Brokerage service provides pairing services between the service requester and the service provider. This pairing enables the selection of best available resources from the service provider for the execution of a specific task.

The pairing process in a brokerage service involves allocation and support functions such as:

- Allocating the appropriate resource or a combination of resources for the task execution.
- Supporting users’ deadline and budget constraints for scheduling optimizations.

### 2.4.4 Schedulers

A grid application that is organized as a collection of jobs is usually designed to have these jobs execute in parallel on different machines in the grid.

Schedulers are responsible for sending a job to a given machine to be executed. More advanced grid systems are including a job scheduler of some kind that automatically finds the most appropriate machine on which to run any given job that is waiting to be executed. Brokerage service is used in place of scheduler.
The user can query the grid system to see how his application and its sub jobs are progressing. If a job is successfully executed then the results are transferred to the specified location and report the job completion status to the user. A job execution may fail due the following reasons:

- **Programming error:** The job stops part way with some program fault.
- **Hardware or Power failure:** The machine or devices being used stop working in some way.
- **Communications interruption:** A communication path to the machine has failed or is overloaded with other data traffic.
- **Excessive slowness:** The job might be in an infinite loop or normal job progress may be limited by another process running at a higher priority or some other form of contention.

In case of job execution failures, schedulers, have to resubmit the jobs either on the same nodes or other nodes for execution.

### 2.5 Overview of some of Available Resource Management Frameworks

We studied four monitoring and information services for grid computing environments: the Globus Toolkit [IBM Red Book, 2003] Monitoring and Discovery Service2 [Czajkowski, K., 2001], the European Data Grid Monitoring Architecture (RGMA) [Tierney, B., R. Avdt, 2002], Hawkeye [Hawkeye], part of Condor [Condor] project and Gridbus Broker [Buyya, VenuGopal, 2004]. The following subsections, briefly discusses about these frameworks.

#### 2.5.1 Monitoring and Discovery Service (MDS)

The Monitoring and Discovery Service (MDS), [Czajkowski, K., 2001], [MDS] is the grid information service used in the Globus Toolkit [Foster, I, C. Kesselman, 1997]. MDS is built on top of the Lightweight Directory Access Protocol (LDAP) [OpenLdap].
MDS is used primarily to address the resource selection problem, namely, how a user identifies the host or set of hosts on which to run an application.

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

1. Grid Resource Information Service (GRIS)
2. Grid Index Information Service (GIIS)
3. Information Provider
4. MDS Client

The following figure 2.3 represents the conceptual view interconnection of the MDS components. The information provider obtains the resource information and it is passed to GRIS. GRIS registers its local information with the GIIS, which also registers with another GIIS, and so on. MDS clients can get the resource information directly from GRIS (for local resources) and/or a GIIS (for grid-wide resources). The MDS uses LDAP, which provides the decentralized maintenance of resource information.

Figure 2.3: MDS Architecture [IBM Red Book, 2003]
GRIS is the repository of local resource information derived from information providers. GRIS is able to register its information with a GIIS, but GRIS itself does not receive registration requests. The local information maintained by GRIS is updated when requested, and cached for a period of time known as the time-to-live (TTL). If no request for the information is received by GRIS, the information will time out and be deleted. If a later request for the information is received, GRIS will call the relevant information provider(s) to retrieve the latest information.

GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIIS. It can be seen as a grid wide information server. GIIS has a hierarchical mechanism, like DNS, and each GIIS has its own name. This means client users can specify the name of a GIIS node to search for information.

2.5.2 RGMA

The Relational Grid Monitoring Architecture (RGMA) [DataGrid, 2002] monitoring system is an implementation of the Grid Monitoring Architecture (GMA) [Tierney, B., R. Avdt, 2002] defined within the Global Grid Forum (GGF) [Global Grid Forum]. It is based on the relational data model [Fisher, S., 2001] and Java Servlet technologies [Java Servlet Technology]. Its main use is the notification of events i.e., a user can subscribe to a flow of data with specific properties directly from a data source. For example, a user can subscribe to a load-data data stream, and create a new Producer/Consumer pairing to allow notification when the load reaches some maximum or minimum.

The following figure 2.4 shows the GMA architecture. GMA is an architecture for monitoring components that specifically addresses the characteristics of grid platforms. GMA consists of three components: Consumers, Producers, and a Registry. Producers register themselves with the Registry, and Consumers query the Registry to find out what types of information are available and to locate the corresponding Producers. Then the Consumer can contact a specific Producer directly. GMA as defined currently does not specify the protocols or the underlying data model to be used.
Figure 2.4: Grid Monitoring Architecture [William E. Johnston, 2002]

2.5.3 Hawkeye

Hawkeye [Hawkeye] is a tool developed by the Condor [Condor] group and designed to automate problem detection, for example to identify high CPU load, high network traffic, or resource failure within a distributed system. Its underlying infrastructure builds on the Condor [Condor] and ClassAd [Raman, R., 2001] technologies. The main use case that Hawkeye was built to address is that of being able to offer monitoring information to anyone interested and to execute actions in response to conditions. It also allows for easier software maintenance within a pool. Hawkeye involves two fundamental ideas: its use of the Condor ClassAd Language to identify resources in a pool, and ClassAd Matchmaking to execute jobs based on attribute values of resources to identify problems in a pool. A ClassAd is a set of attribute/value pairs (e.g., “operating system” and “Linux”). The Manager performs ClassAd Matchmaking between a Trigger ClassAd, submitted by a client, and all Startd ClassAds. A Trigger ClassAd specifies an event and a job to execute if the event occurs. For example, consider the case in which a Trigger ClassAd specifies an event in which the CPU load is greater than 50 and a job that will kill a Netscape client running on the
matched machine; if any machine advertises a Startd ClassAd with a CPU load value of greater than 50, the Manager will kill that machine’s Netscape process. The following figure 2.5 shows the architecture of Hawkeye.

![Hawkeye Architecture](image)

**Figure 2.5: Hawkeye Architecture [Xuehai, Jeffrey, Jennifer, 2003]**

The architecture of Hawkeye comprises four major components: Pool, Manager, Monitoring Agent, and Module. The components are organized in a four-level hierarchical structure. A pool is a set of computers, in which one computer serves as the Manager and the remaining computers serve as Monitoring Agents. A Manager is the head computer in the pool that collects and stores (in an indexed resident database) monitoring information from each agent registered to it. It is also the central target for queries about the status of any pool member. A Monitoring Agent is a distributed information service component that collects ClassAds from each of its Modules and then integrates them into a single *Startd ClassAd*. At fixed intervals, the agent sends the *Startd ClassAd* to its registered Manager. An agent can also directly answer queries about a particular Module; however, the client must first consult the Manager for the
agent's IP address. A Module is simply a sensor that advertises resource information in a ClassAd format.

### 2.5.4 Gridbus Broker

The Gridbus Broker [Srikumar, Rajkumar 2004] makes scheduling decisions on where to place the jobs on the grid depending on the computational resources characteristics (such as availability, capability, and cost), the user's Quality-of-Service requirements such as the deadline and budget, and the proximity of the required data or its replicas to the computational resources. Catalogues of replicated data describe the size of the file, the location of the file, the date it was produced, the number of events and other such attributes. Given a job and the input file(s) it requires, the broker looks up the Replica Catalogue at the local site to locate the sites where the required input file is and its size. Then it takes into account various other factors such as the cost, the computing power available at the site, the network bandwidth, the resource reputation, and the account information to make a decision on where to dispatch the job. The broker identifies resource service prices by querying the Grid Market Directory (GMD).

If an application needs to access remote databases, Gridbus provides transparent data access mechanisms and a catalogue that supports logical mapping of data files to a distributed storage devices. The broker performs discovery and online extraction of data-sets from the closest data sources and then farms out analysis jobs to optimal resources. The brokers evaluates whether to process jobs on a resource where the data is available by moving the application code or move data to a resource where the application is available, or move both of them to a suitable computing resource.

The architecture of the Gridbus broker is shown in figure 2.6. The inputs to the broker are the tasks and the associated parameters with their values. These can be specified within a “plan” file that specifies the tasks and the types of the parameters and their values for these tasks.
A task is a sequence of commands that describe the user’s requirements. For example, the user may specify an application to be executed at the remote site, by providing an input file to be copied over before execution and the results to be returned back after required execution. A task encapsulates this information within its description. A task is accompanied by parameters which can either be static or dynamic. A static parameter is a variable whose domain is well-defined either as a range of values, as a single static value or as one among a set of values. A dynamic parameter has either an undefined or an unbounded domain whose definition or boundary conditions respectively, are required to be established at runtime. As an example, in the current implementation, a parameter type has been defined which describes a set of files over which the application has to be executed. This set can be described as a wildcard search within a physical or a logical directory, to be resolved at runtime, thus creating a dynamic parameter.

The task requirements drive the discovery of resources such as computational nodes and data resources. The resource discovery module gathers information from
remote information services such as the Grid Index Information Service (GIIS) [Czajkowski, K., 2001] for availability of compute resources. Optionally, the list of available compute resources can be provided by the user to the broker. The broker also interacts with the information service on each computational node to obtain its properties. Data files can be organized as Logical File Names (LFNs) within a virtual directory structure using a Replica/Data Service Catalog. Each LFN maps to one or many Physical File Names (PFNs) somewhere on the grid, usually specified via URLs. The broker will resolve the LFNs to the appropriate physical file location(s) by querying the catalog.

The task description, i.e. the task along with its associated parameters, is resolved or "decomposed" into jobs. A job is an instantiation of the task with a unique combination of parameter values. It is also the unit of work that is sent to a grid node. The set of jobs along with the set of service nodes are an input to the scheduler. For jobs requiring remote data, the scheduler interacts with a network monitoring service to obtain the information about current available bandwidth between the data sources and the compute resources. In the current implementation, the Network Weather Service (NWS) [Wolski, R] has been used to obtain this information.

The jobs are dispatched to the remote node through the Actuator component. The Actuator submits the job to the remote node using the functionality provided by the middleware running on it. The Actuator has been designed to operate with different grid middleware frameworks and toolkits such as Globus 2.4 [Foster, I, C. Kesselman, 1997]. The task commands are encapsulated within an agent who is dispatched to and executed on the remote machine. If a data file has been associated with the job and a suitable data host identified for that file, then the Agent obtains the file through a remote data transfer from the data host. Additionally, it may require some configuration or input parameter files that it obtains from the broker through a mechanism such as a Globus Access to Secondary Storage (GASS) server [Bester J, Foster I, 1999]. These files are assumed to be small and in tens or hundreds of kilobytes which impact the overall execution time of a job negligibly whereas the data files are in the range of megabytes or larger. On the completion of execution, the agent returns any results to the broker and provides debugging information. The monitoring component keeps track of job status –
whether the jobs are queued, executing, finished successfully or failed. The bookkeeper keeps a persistent record of job and resource states throughout the entire execution.

2.6 Comparison of GMIS with MDS, RGMA, Hawkeye and Gridbus Broker

In the study of resource management services discussed above, we found that each approach has different behaviors, often due to their different design goals. To take advantage of these approaches, we included all features of these services in our Grid Management Information Server (GMIS) framework.

The following table 2.2 shows the comparison among MDS2, R-GMA, Hawkeye, GridBus Broker and GMIS framework. Following parameters are used to compare these along with the questions and design considerations mentioned in Chapter 1, Section 1.6.

- **Primary objective**: The capabilities provided or problems solved by the frameworks.
- **Decentralized**: Support for distributed deployment of framework and providing a single user interface for all the activities.
- **Dynamic Grid Network Discovery**: Capability to discover and utilize all the available grids, clusters, nodes and applications automatically and maintain hierarchical information.
- **Dynamic Event based Discovery**: Capability to discover and utilize a newly available resource such as a node or application based on an event sent by the resource to the framework.
- **User based Discovery**: Capability to discovery and utilize grid resource based on the input provided by the user.
- **Push Model**: Information about the various resources and jobs is updated based on the events sent by the resources.
- **Pull Model**: Information about the various resources and jobs is updated based on the periodic polling the resources and parsing the responses.
- **Data Server**: Centralized repository of all the resource and jobs information.
- **Directory Server**: Service provider that maintains the hierarchical information of the resources and provide information as needed by other services or middleware.
• **Multi selection capability:** Based on the job requirements, the user will be provided with a list of available resources and user has a choice to select a particular resource for job execution.

• **Data subscription support:** Any application or middleware can subscribe for existing resource or job information and also any changes to them.

• **Resource Mining:** Capacity to display resource information based on historical data about job execution and resource capabilities.

• **MDO Support:** Support for Multidisciplinary Design Optimization, which is used in design of complex engineering systems by performing system level analysis.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>MDS2</th>
<th>RGMA</th>
<th>Hawkeye</th>
<th>Gridbus Broker</th>
<th>GMIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary objective</td>
<td>Brokerage</td>
<td>Notification of events &amp; Monitoring</td>
<td>Automate problem detection &amp; Monitoring</td>
<td>Brokerage &amp; Scheduling</td>
<td>Discovery, Monitoring &amp; Brokerage</td>
</tr>
<tr>
<td>Decentralized</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Grid Network Discovery</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Event based Discovery</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>User based discovery</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Push model</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Pull model</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Data Server</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Directory Server</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi selection capability</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data subscription support</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Resource mining</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MDO support</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How to know where the resources are?</td>
<td>Information provider</td>
<td>Notification of events</td>
<td>Collector</td>
<td>Grid Information Service</td>
<td>Dynamic network discovery, Dynamic Event based discovery and user based discovery.</td>
</tr>
<tr>
<td>How to identify resources?</td>
<td>GIIS and GRIS</td>
<td>Registry</td>
<td>Class Ad Objects</td>
<td>Grid Information Service</td>
<td>Database and Topology manager MIT</td>
</tr>
<tr>
<td>How to get permissions to use them?</td>
<td>Information providers</td>
<td>Producers Manager and Agent</td>
<td>Grid Resource Broker</td>
<td>Agent Software</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------</td>
<td>-----------------------------</td>
<td>---------------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>How to know what are the applications available in the resources?</td>
<td>Not addressed</td>
<td>Not addressed</td>
<td>Not addressed</td>
<td>Data Catalogue</td>
<td>Agent software</td>
</tr>
<tr>
<td>How to use the resources?</td>
<td>GRIS and GIIS</td>
<td>Consumers Manager Grid Resource Broker</td>
<td>Grid Resource Broker</td>
<td>Resource Selector</td>
<td></td>
</tr>
<tr>
<td>How to submit remote jobs?</td>
<td>Middleware</td>
<td>Middleware</td>
<td>Middleware</td>
<td>Grid Resource Broker</td>
<td>Using user interface provided in Viewer and as well as Middleware.</td>
</tr>
<tr>
<td>How to get access to resources to all the machines simultaneously?</td>
<td>GRIS and GIIS</td>
<td>Producers Agents Grid Resource Broker</td>
<td>Grid Resource Broker</td>
<td>Resource Selector</td>
<td></td>
</tr>
<tr>
<td>What happens if a resource fails?</td>
<td>Not addressed</td>
<td>Not addressed</td>
<td>Not addressed</td>
<td>Addressed</td>
<td>Addressed</td>
</tr>
<tr>
<td>How input/output files are managed?</td>
<td>GridFTP</td>
<td>Grid Sandbox Database tables</td>
<td>GridFTP</td>
<td>FTP</td>
<td></td>
</tr>
<tr>
<td>Seamless and transparent user access to resource information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Easy to use web-based interface</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Collation of information from multiple resources</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Efficient resource management</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scalable and Adaptable framework</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Poll and Event based information gathering mechanisms</td>
<td>No</td>
<td>Partially (event based)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of MDS2, R·GMA, Hawkeye, Gridbus Broker and GMIS
2.7 Summary

In this chapter we presented an overview of grid computing, ancestors of the Grid and OGSA architecture of grid computing. It discussed about basic grid technologies like Globus Toolkit, IBM Grid Tool box, SUN Grid Engine, Avaki, and Microsoft .NET technologies. Comparison among these technologies is presented. It extensively discussed about grid activities of Global Grid Forum, Condor, European DataGrid, PRAGMA and GridBus.

It discussed about the resource management and various resource specific characterization capabilities. It presented a detail literature survey on some of the popular resource management frameworks MDS2, R-GMA, Hawkeye and GridBus Broker. Finally, comparison of these frameworks with proposed GMIS framework is presented. Several features not available in the other popular resource management frameworks are also been discussed. Next chapter will discuss about the GMIS Framework architecture.