Chapter 2

Grid Technologies and Resource Management Systems

This chapter presents an overview of Grid technologies with major emphasis on resource management and scheduling systems. It discusses some of the important technological advances that have led to the emergence of Grid computing. It presents the taxonomy of Grid resource management systems, briefly followed by a survey of some representative example systems.

2.1 Introduction

The last decade has seen a substantial increase in commodity computer and network performance, mainly as a result of faster hardware and more sophisticated software. Nevertheless, there are still problems in the fields of science, engineering, and business, which cannot be effectively dealt with using the current generation of supercomputers. In fact, due to their size and complexity, these problems are often resource (computational and data) intensive and consequently entail the use of a variety of heterogeneous resources that are not available in a single organisation.

The ubiquity of the Internet as well as the availability of powerful computers and high-speed network technologies as low-cost commodity components is rapidly changing the computing landscape and society. These technology opportunities have led to the possibility of using wide-area distributed computers for solving large-scale problems, leading to what is popularly known as Grid computing [48]. The term Grid is chosen as an analogy to the electric power Grid that provides consistent, pervasive, dependable, transparent access to electricity, irrespective of its source. Such an approach to network computing is known by several names: metacomputing, scalable computing, global computing, Internet computing, and more recently Peer-to-Peer (P2P) computing [4].

Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices (see Figure 2.1) that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce.

The concept of Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high throughput computing, distributed supercomputing, and service-oriented computing. Moreover, due to the rapid growth of the Internet and Web, there has been a growing interest in Web-based distributed computing, and many projects have been started and aim to exploit the Web as an infrastructure for running coarse-grained distributed and parallel applications. In this context, the Web has the capability to act as a platform for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous Grid-based infrastructure.

Grid applications (typically multi-disciplinary and large-scale processing applications) often couple resources that cannot be replicated at a single site, or may be globally located for other practical reasons (see Figure 2.1). These are some of the driving forces behind the foundation of global Grids. In this light, the Grid allows users to solve larger-scale problems by pooling together resources that could not be coupled easily before. Hence, the Grid is not only a computing infrastructure, for large applications, it is a technology that can bond and unify remote and diverse distributed resources ranging from meteorological sensors to data vaults, and from parallel supercomputers to personal digital organizers. As such, it will

provide pervasive services to all users that need them.

Figure 2.1: Towards Grid computing: A conceptual view.

A Grid can be viewed as a seamless, integrated computational and collaborative environment and a high level view of activities within the Grid. The users interact with the Grid resource broker for solving problems, which in turn performs resource discovery, scheduling, and processing application jobs on the distributed Grid resources.

2.2 Major Technological Milestones: Enabling Grid and P2P Computing

The major technological advancements from 1960 to date in computing and networking technologies that led to the emergence of P2P and Grid computing is shown in Figure 2.2. There has been the rise and fall of different systems. In 1960, mainframes mainly from IBM were serving the needs of computing users, but a decade later DEC introduced less expensive minicomputers that took over mainframes market share. During the 1980s, vector computers (e.g., Crays) and later parallel computers (e.g., MPP systems) were serving the needs of grand challenging applications. We briefly discuss technological milestones in networking followed by computing.

The communication infrastructure for computational Grids is the Internet, that began as a modest research network, supported by the Advanced Research Projects Agency (ARPA) of the US Defense Department. The ARPA's effort started as a response to the USSR's launch of Sputnik, the first artificial earth satellite in 1957 [119]. The ARPANET with four nodes was first established in 1969 at the University of California at Los Angeles, Stanford Research Institute, University of California Santa Barbara (UCSB), and University of Utah during the September, October, November, and December months respectively. By the mid-1970s, the ARPANET *Internet* work embraced more than 30 universities, military sites and government contractors and its user base expanded to include the larger computer science research community. Bob Metcalfe's Harvard PhD Thesis outlines the idea for the Ethernet in 1973 that came into existence in 1976 [114]. Vint Cerf and Bob Kahn proposed the Transmission Control Program (TCP) in 1974, which was split into TCP/IP in 1978. By 1983, the network still consisted of a network of several hundred computers on only a few local area networks. In 1985, the National Science Foundation (NSF) arranged with ARPA to support a collaboration of supercomputing centers and computer science researchers across the ARPANET. In 1989, responsibility and management for the ARPANET, was officially passed from military interests to the academically oriented NSF. Much of the Internet's etiquette and rules for behavior were established during this time. The Internet Engineering Task Force (IETF) was formed during 1986 as a loosely self-organized group of people who contribute to the engineering and evolution of Internet technologies [123].

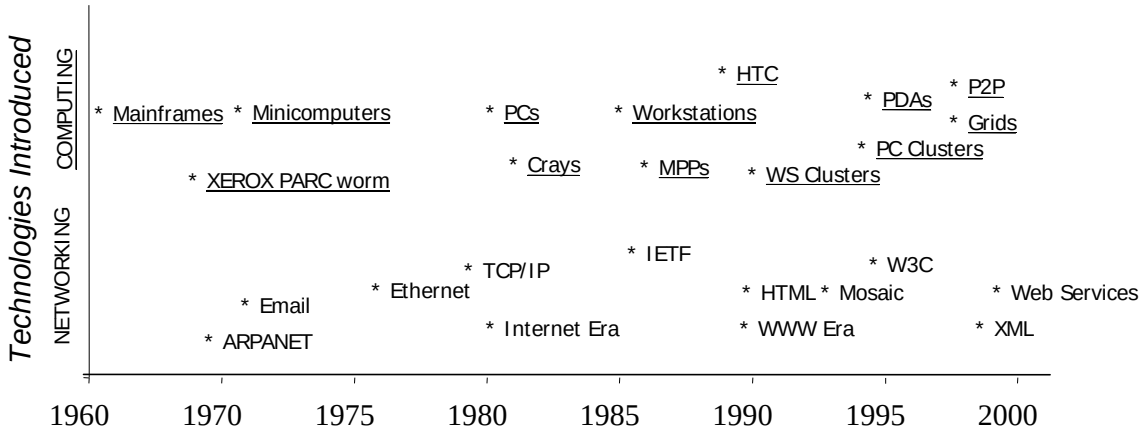

Figure 2.2: Major milestones in networking and computing technologies from the year 1960 onwards.

The invention of the Web [129] in 1989 by Tim Berners-Lee of CERN, Switzerland, for sharing information with ease has fueled a major revolution in computing. It provided the means for creating and organizing documents (using HTML language) with links and accessing them online transparently, irrespective of their location (using http protocols, browsers, and servers). The World-Wide Web consortium (W3C) [143] formed in 1994 is engaged in developing new standards for information interchange such as XML (eXtended Markup Language) Web services for providing remote access to software and applications as a service.

In the early 1970s when computers were first linked by networks, the idea of harnessing unused CPU cycles was born [136]. A few early experiments with distributed computing—including a pair of programs called *Creeper and Reaper*—ran on the Internet's predecessor, the ARPAnet. In 1973, the Xerox Palo Alto Research Center (PARC) installed the first Ethernet network and the first fully-fledged distributed computing effort was underway. Scientists at PARC developed a program called “worm” that routinely cruised about 100 Ethernet-connected computers. They envisioned their worm migrating from one machine to another to harness idle resources for beneficial purposes. The worm would roam throughout the PARC network, replicating itself in each machine's memory. Each worm used idle resources to perform a computation and had the ability to reproduce and transmit clones to other nodes of the network. With the worms, developers distributed graphic images and shared computations for rendering realistic computer graphics.

Since 1990, with the maturation and ubiquity of the Internet and Web technologies along with the availability of powerful computers and system area networks as commodity components, distributed computing scaled to a new global level. The availability of powerful PCs and workstations, and high-speed networks (e.g., Gigabit Ethernet) as commodity components has led to the emergence of clusters [92] serving the needs of high performance computing (HPC) users. The ubiquity of the Internet and Web technologies along with the availability of many low-cost and high-performance commodity clusters within many organizations has prompted the exploration of aggregating distributed resources for solving large scale problems of multi-institutional interest. This has led to the emergence of computational Grids and P2P networks for sharing distributed resources. The Grid community is generally focused on aggregation of distributed high-end machines such as clusters, whereas the P2P community (e.g., SETI@Home [141]) is looking into sharing low-end systems, such as PCs connected to the Internet and contents (e.g., exchange music files via Napster and Gnutella networks). Given the number of projects and forums [74][91] started all over the world in early 2000, it is clear that interest in the research, development, and deployment of Grid and P2P computing technologies, tools, and applications is rapidly growing.

Already application domains like Monte Carlo simulations and parameter sweep applications (e.g., ionization chamber calibration [19], drug design [106], operations research, electronic CAD, and ecological modeling), where large processing problems can easily be divided into sub-problems and solved independently, are taking great advantage of Grid computing.

2.3 Grid Computing Environments

2.3.1 Resource Management Challenges

The Grid environment contains heterogeneous resources, local management systems (single system image OS, queuing systems, etc.) and policies, and applications (scientific, engineering, and commercial) with varied requirements (CPU, I/O, memory, and/or network intensive). The *producers* (also called resource owners) and *consumers* (who are the users) have different goals, objectives, strategies, and demand patterns [99]. More importantly, both resources and end-users are geographically distributed with different time zones. A number of approaches for resource management architectures have been proposed and the prominent ones are: centralized, decentralized, and hierarchical.

In managing the complexities present in large-scale Grid-like systems, traditional approaches are not suitable as they attempt to optimize system-wide measures of performance. Traditional approaches use centralized policies that need complete state information and a common resource management policy, or decentralized consensus based policy. Due to the complexity in constructing successful Grid environments, it is impossible to define an acceptable system-wide performance matrix and common fabric management policy. Therefore, hierarchical and decentralized approaches are suitable for Grid resource and operational management [99]. Within these approaches, there exist different economic models for management and regulation of supply-and-demand for resources [103]. The Grid resource broker mediates between producers and consumers (see Figure 2.3). The resources are Grid enabled by deploying low-level middleware systems on them. The core middleware deployed on producer's Grid resources supports the ability to handle resource access authorization and permits only authorized users to access them. The user-level and core middleware on consumer's resources supports the ability to create Grid enabled applications or necessary tools to support the execution of legacy applications on the Grid. Upon authenticating to the Grid, consumers interact with resource brokers for executing their applications on remote resources. The resource broker takes care of resource discovery, selection, aggregation, data and program transportation, initiating execution on remote resources and gathering results.

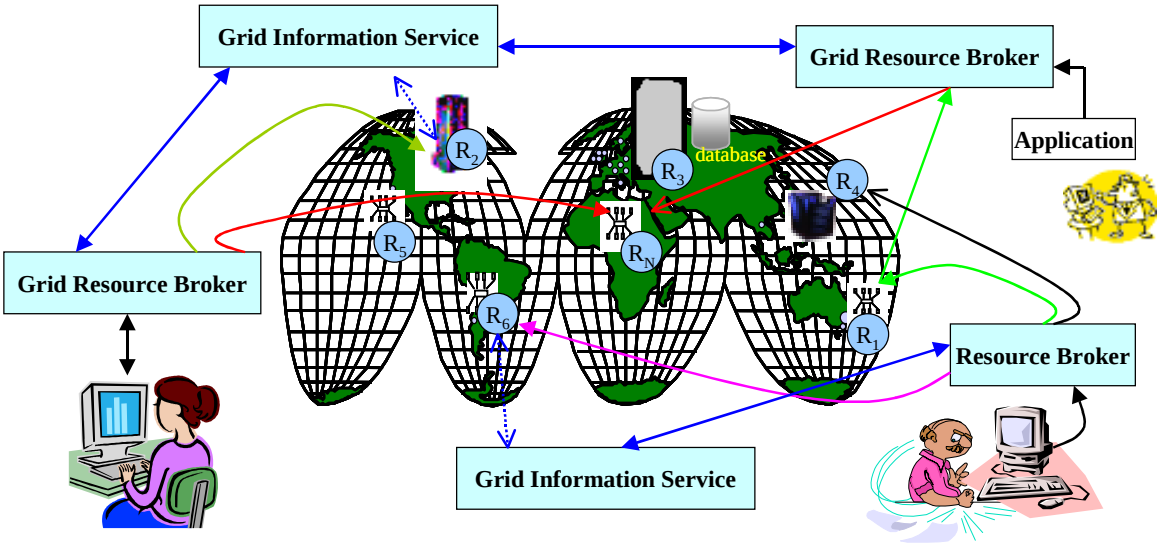

Figure 2.3: A high-level view of the Grid and interaction between its entities.

For the operation of a computational Grid, the broker discovers properties of resources that the user can access through the Grid information server(s), negotiates with (Grid-enabled) resources or their agents using middleware services, maps tasks to resources (scheduling), stages the application and data for processing (deployment) and finally gathers results [100]. It is also responsible for monitoring application execution progress along with managing changes in the Grid infrastructure and resource failures. There are a number of projects, worldwide, actively exploring the development of various Grid computing system components, services, and applications.

2.3.2 Grid Components

This section briefly highlights some of the general principles that underlie the construction of the Grid. In particular, the idealized design features that are required by a Grid to provide users with a seamless computing environment are discussed. Four main aspects characterise a Grid:

- *Multiple Administrative Domains and Autonomy:* Grid resources are geographically distributed across multiple administrative domains and owned by different organizations. The autonomy of resource owners needs to be honored along with their local resource management and usage policies.
- *Heterogeneity*: A Grid involves a multiplicity of resources that are heterogeneous in nature and will encompass a vast range of technologies.
- *Scalability*: A Grid might grow from a few integrated resources to millions. This raises the problem of potential performance degradation. Consequently, applications that require a large number of geographically located resources must be designed to be latency and bandwidth tolerant.
- *Dynamicity or Adaptability*: In a Grid, resource failure is the rule rather than the exception. In fact, with so many resources in a Grid, the probability of some resource failing is high. Resource managers or applications must tailor their behavior dynamically and use the available resources and services efficiently and effectively.

The steps necessary to realize a Grid include:

- The integration of individual software and hardware components into a combined networked resource (e.g., a single system image cluster).
- The deployment of:
    - Low-level middleware to provide a secure and transparent access to resources.
    - User-level middleware and tools for application development and the aggregation of distributed resources.
- The development and optimization of distributed applications to take advantage of the available resources and infrastructure.

The Grid is made up of a number of components from enabling resources to end user applications. A layered architecture of the Grid is shown in Figure 2.4. The key components of a Grid are:

- **Grid Fabric:** This consists of all the globally distributed resources that are accessible from anywhere on the Internet. These resources could be computers (such as PCs, SMPs, clusters) running a variety of operating systems (such as UNIX or Windows), as well as resource management systems such as LSF (Load Sharing Facility), Condor, PBS (Portable Batch System) or SGE (Sun Grid Engine), storage devices, databases, and special scientific instruments such as a radio telescope or particular heat sensor.
- **Core Grid Middleware:** This offers core services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading.
- **User-Level Grid Middleware:** This includes application development environments, programming tools, and resource brokers for managing resources and scheduling application tasks for execution on global resources.
- **Grid Applications and Portals:** Grid applications are typically developed using Grid-enabled languages and utilities such as MPI (message-passing interface) or Nimrod parameter specification language. An example application, such as a parameter simulation or a grand-challenge problem, would require computational power, access to remote data sets, and may need to interact with scientific instruments. Grid portals offer Web-enabled application services, where the users can submit and collect results for their jobs on remote resources through the Web.

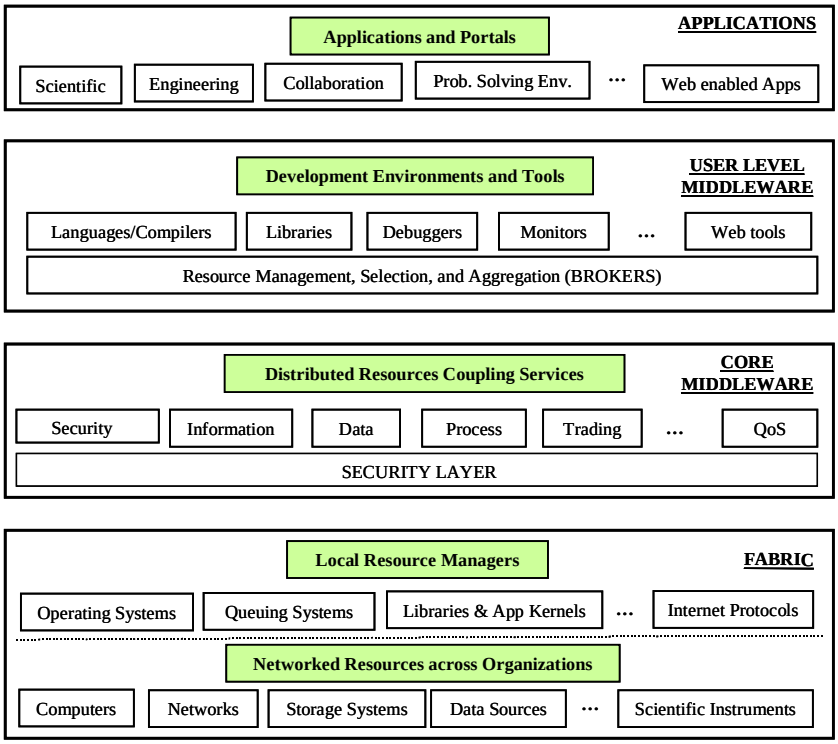

Figure 2.4: A layered Grid architecture and components.

2.3.3 Grid Computing Projects

There are many international Grid projects worldwide, which are hierarchically categorized as integrated Grid systems, core middleware, user-level middleware, and applications driven efforts (see Table 1). A listing of the majority of projects in Grid computing worldwide along with pointers to their websites can be found in [91][72]. Selected projects are further grouped into country/continents and discussed in [74].

Table 1: Hierarchical organization of major Grid efforts.

Category	Project	Organisation	Remarks
Integrated Grid Systems	NetSolve	U. Tennessee	A programming and runtime system for accessing high-performance libraries and resources transparently.
	Ninf	AIST, Japan	Functionality is similar to NetSolve.
	ST-ORM	UPC, Barcelona	A scheduler for distributed batch systems.
	MOL	Paderborn U.	A scheduler for distributed batch systems.
	Albatross	Vrije U.	Object oriented programming system.
	PUNCH	Purdue U.	A portal computing environment and service for applications.
	Javelin	UCSB	Java-based programming and runtime system.
	XtremWeb	Paris-Sud U.	A global computing environment
	MILAN	Arizona and NY	Aims to provide end-to-end services for transparent utilization and management of networked resources
	DISCWorld	U. of Adelaide	A distributed information-processing environment.
	Unicore	Germany	Java-based environment for accessing remote supercomputers.

Core Middleware	Cosm	Mithral	A toolkit building P2P applications.
	Globus	ANL and ISI	Globus provides uniform and secure environment for accessing remote computational and storage resources.
	GRACE	Monash U.	A distributed computational economy framework for service oriented Grid computing.
	GridSim	Monash U.	A toolkit for Grid simulation.
	JXTA	Sun Microsystems	A Java-based framework and infrastructure for P2P computing.
	Legion	U. of Virginia	A Grid operating system providing transparent access to distributed resources.
	P2P Accelerator	Intel	A basic infrastructure for creating P2P applications for .NET platform.
User-level Middleware: *Schedulers*	AppLeS	UCSD	Application specific scheduler.
	Condor-G	U. of Wisconsin	A wide area job processing system.
	Nimrod-G	Monash U.	Economic-based Grid resource broker for parameter sweep/task farming applications.
User-level Middleware: *Programming Environments*	MPICH-G	Northern Illinois U.	MPI implementation on Globus.
	Nimrod parameter programming tools	Monash U.	A declarative language parametric programming.
	MetaMPICH	RWTH, Aachen	MPI programming and runtime environment.
	Cactus	Max Planck Institute for Gravitational Physics	A framework for writing parallel applications. It is developed using the MPICH-G and Globus.
	GrADS	Rice U.	Grid application development tools.
	GridPort	SDSC	Tools for creating computing portals.
Applications and application driven Grid efforts	European Data Grid	CERN	High Energy Physics, Earth Observation, Biology
	GriPhyN	UCF and ANL	High Energy Physics
	PPDG	Caltech and ANL	High Energy Physics
	Virtual Laboratory	Monash U and WEHI	Molecular modeling for drug design
	HEPGrid	Melbourne U	High Energy Physics applications
	NEESGrid	NCSA	Earthquake Engineering
	Geodise	Southampton U.	Aerospace Design Optimisation
	Fusion Grid	Princeton/ANL/	Magnetic fusion
	IPG	NASA	Aerospace
	Active Sheets	Monash, QUT, & DSTC	Spread sheet processing

	Earth System Grid	LLNL, ANL, &NCAR	Climate Modeling
	Virtual Instruments	UCSD	Neuroscience
	National Virtual Observatory	Johns Hopkins U. & Caltech	Access to distributed astronomical databases and processing.

Grids can be used to solve grand challenge problems in areas such as biophysics, chemistry, biology, scientific instrumentation [19], drug design **[110]**, tomography [127], high energy physics **[64]**, data mining, financial analysis, nuclear simulations, material science, chemical engineering, environmental studies, climate modeling **[6]**, weather prediction, molecular biology, structural analysis, mechanical CAD/CAM and astrophysics. Although wide-area distributed supercomputing has been a popular application of the Grid, there are a large number of other applications that can benefit from the Grid **[140] [75]**.

2.4 Resource Management Systems Taxonomy

Depending on the focus and application target, the Grid Resource Management Systems (RMSs) are broadly classified into Computational Grids, Data Grids, and Service Grids. In [65], a taxonomy for Grid resource management systems is developed, which classifies resource management systems by characterizing different attributes as summarized in Table 2.2. The taxonomy focuses on the type of Grid system, machine organization, resource model characterization, and scheduling characterization.

Table 2.2: Taxonomy of Grid resource management systems.

Attributes of Resource Management Systems	Taxonomy
Grid Type (Service focus)	Computational Grids, Data Grids, Service Grids
Machine organization	Flat, cell (flat cells and hierarchical cells), hierarchical
Resource model	Schema, Object model (fixed or extensible)
Namespace organization	Relational, Hierarchical, Graph
QoS	Soft, Hard, None
Resource Information Store	Network Directory and Distributed Objects
Resource discovery	Query and Agents
Resource Info Dissemination	Batch/Period (push or pull), Online/On-demand
Scheduler organization	Centralised, Hierarchical, Decentralised
Scheduling policy	System-Centric, User Centric
State estimation	Predictive (Heuristics, Pricing models, machine learning) and Non-Predictive
Rescheduling	Periodic, Event Driven

The organization of the machines in the Grid affects the communication patterns of the RMS and thus determines the scalability of the resultant architecture. In a flat organization all machines can directly communicate with each other. In a hierarchal organization machines at the same level can directly communicate with the machines directly above them or below them, or peer to them in the hierarchy. The fan out below a machine in the hierarchy is not relevant to the classification. Most current Grid systems use this organization since it has proven scalability. In a cell structure, the machines within the cell communicate between themselves using a flat organization. Designated machines within the cell function

as boundary elements that are responsible for all communication outside the cell (e.g., a cluster with a master node directly accessible from outside and manages internal nodes).

The resource model determines how applications and the RMS describe and manage Grid resources. In a schema based approach the data that comprises a resource is described in a description language along with some integrity constraints (e.g., Condor ClassAd). In the object model extensible approach the resource model provides a mechanism to extend the definition of the object model managed by the RMS (e.g., Legion object model [2]). The resource namespace influences the design of the resource management protocols and affects the discovery methods. The quality of service (QoS) allows users to specify the level of service they are expecting from a resource and an RMS. The resource information store, which is updated based on the dissemination model, provides Grid information services. The brokers can discover resources by querying the information store.

The Grid scheduling systems can be classified into centralized, hierarchical, and decentralized. As the resource availability in the Grid changes with time, the scheduling systems need to be adaptive. This is achieved by evaluating the current schedule (state estimation) based on predictive techniques, and then developing a new schedule (rescheduling) to meet the users requirements. The re-scheduling can be initiated periodically or whenever some event occurs (e.g., a notification of job completion).

2.5 Mapping Taxonomy to Some Grid Resource Management Systems

There are many existing Grid computing projects currently underway. They include Globus, Legion, NetSolve, AppLeS, and Condor. This section provides a brief description of each system and then classifies the resource management system attributes according to our taxonomy. A summary of architectural design choices made by a few popular Grid resource management systems is shown in Table 2.3. The Nimrod-G resource broker, developed in this thesis, has been included to facilitate an effective comparison of its features and design choices with related systems.

Table 2.3: Grid resource management systems and their architecture choices.

System	Grid Type	Organization	Resource: model, namespace, QoS, information store, discovery, dissemination.	Scheduling: organisation, state-estimation, rescheduling, and policy.
AppLeS	Computational Grid (scheduling)	Hierarchical	Uses resource model provided by the underlying Globus, Legion, or NetSolve middleware services	Decentralized scheduler, predictive heuristic state estimation, online rescheduling, fixed application oriented policy (system-centric)
DataGrid	Data Grid Computational Grid	Hierarchical	Extensible schema model, hierarchical namespace, no QoS, LDAP network directory store, distributed query-based discovery, periodic push dissemination.	Hierarchical schedulers, predictive heuristic state estimation, online rescheduling, extensible scheduling policy
Condor	Computational Grid	Flat	Extensible schema model, hybrid namespace, no QoS, other network directory store, centralized query based discovery, periodic push dissemination	Cooperative/Centralized scheduler
Globus	Grid Toolkit	Hierarchical Cells	Extensible schema model, hierarchical namespace, soft QoS, LDAP network directory store, distributed query based discovery, periodic push dissemination	Hierarchical scheduler, ad-hoc extensible policy
Javelin	Computational Grid	Hierarchical	Fixed object model, graph namespace, soft QoS, other network directory store, distributed query based discovery, periodic push dissemination	Decentralized scheduler, fixed application oriented policy
Legion	Computational Grid	Flat Hierarchical	Extensible object model, graph namespace, soft QoS, object model store, distributed query-based discovery, periodic pull dissemination.	Hierarchical scheduler, ad-hoc extensible scheduling policies
MOL	Computational Grid	Hierarchical Cells	Extensible schema model, hierarchical namespace, no QoS, object model store,	Decentralized scheduler, extensible ad-hoc scheduling

			distributed query based discovery, periodic push dissemination	policies
NetSolve	Computational & Service Grid	Hierarchical	Extensible schema model, hierarchical namespace, soft QoS, centralized query-based discovery, periodic push dissemination.	Decentralized scheduler, fixed application oriented policy
Ninf	Computational & Service Grid	Hierarchical	Extensible schema model, relational namespace, no QoS, centralized query based resource discovery, periodic push for dissemination.	Decentralized scheduler
PUNCH	Computational & Service Grid	Hierarchical	Extensible schema model, hybrid namespace, soft QoS, distributed query-based discovery, periodic push dissemination.	Decentralized scheduler, machine learning, fixed system oriented policy
Nimrod-G	Computational & Service Grid	Hierarchical Cells	Uses resource model provided by the underlying Globus or Legion middleware services and extends with computational economy approach	Decentralized scheduler, predictive pricing models, event driven rescheduling, fixed application oriented scheduling policy

2.5.1 AppLeS: A Network Enabled Scheduler

The AppLeS [28] (Application Level Scheduling) project at the University of California, San Diego primarily focuses on developing scheduling agents for individual applications on production computational Grids. It uses the services of Network Weather Service (NWS) to monitor changes in performance of resources dynamically. AppLeS agents use static and dynamic application and system information while selecting a viable set of resources and resource configurations. It interacts with other resource management systems such as Globus, Legion, and NetSolve to implement application tasks. The applications have embedded AppLeS agents and thus become self-schedulable on the Grid. The concept of AppLeS has been applied to many application areas including Magnetohydrodynamics [44], Gene Sequence Comparison, and Tomography [127].

Another effort within AppLeS project framework is the development of AppLeS templates. It is similar to Nimrod-G framework and resource broker, but it does not support quality of services-driven scheduling since it does not take Grid economy into consideration.

As the focus of AppLeS project is on scheduling, it follows the resource management model supported by the underlying Grid middleware systems. An AppLeS scheduler is central to the application that performs mapping of jobs to resources, but the local resource schedulers perform the actual execution of application units similar to Nimrod-G. AppLeS schedulers do not offer QoS support and build on a resource model supported by an underlying system. AppLeS can be considered to have a predictive heuristic state estimation model with online rescheduling and application oriented scheduling policies.

2.5.2 Condor: Cycle Stealing Technology for High Throughput Computing

Condor [79][54] is a high-throughput computing environment developed at the University of Wisconsin at Madison, USA. It can manage a large collection of computers such as PCs, workstations, and clusters that are owned by different individuals. Although it is popularly known for harnessing idle computers CPU cycles (cycle stealing), it can be configured to share resources. The Condor environment follows a layered architecture and offers powerful and flexible resource management services for sequential and parallel applications. The Condor system pays special attention to the computer owner's rights and allocates their resources to the Condor pool as per the usage conditions defined by resource owners. Through its unique remote system call capabilities, Condor preserves the job's originating machine environment on the execution machine, even if the originating and execution machines do not share a common file system and/or user ID scheme. Condor jobs with a single process are automatically check-pointed and migrated between workstations as needed to ensure eventual completion. The Condor has been extended to support submission of jobs to resources Grid-enabled using Globus services [57].

Condor can have multiple Condor pools and each pool follows a flat machine organization. The Condor *collector*, which provides the resource information store, listens for advertisements of resource availability.

A Condor resource agent runs on each machine periodically advertising its services to the collector. Customer agents advertise their requests for resources to the collector. The Condor matchmaker queries the collector for resource discovery that it uses to determine compatible resource requests and offers. The agents are then notified of their compatibility. The compatible agents then contact each other directly and, if they are satisfied, then the customer agent initiates computation on the resource.

Resource requests and offers are described in the Condor classified advertisement (ClassAd) language [115]. ClassAds use a semi-structured data model for resource description. Thus, no specific schema is required by the matchmaker allowing it to work naturally in a heterogeneous environment. The ClassAd language includes a query language as part of the data model, allowing advertising agents to specify their compatibility by including constraints in their resource offers and requests.

The matchmaker performs scheduling in a Condor pool. The matchmaker is responsible for initiating contact between compatible agents. Customer agents may advertise resource requests to multiple pools with a mechanism called flocking, allowing a computation to utilize resources distributed across different Condor pools.

The Condor system has recently been enhanced to support creation of personal condor pools. It allows the user to include their Globus-enabled nodes into the Condor pool to create a "personal condor" pool along with public condor pool nodes. The Grid nodes that are included in a personal condor pool are only accessible to the user who created the pool.

Condor can be considered as a computational Grid with a flat organization. It uses an extensible schema with a hybrid namespace. It has no QoS support and the information store is a network directory that does not use X.500/LDAP technology. Resource discovery is achieved through centralized queries with periodic push dissemination. The scheduler is centralized.

2.5.3 Data Grid

CERN, the European Organization for Nuclear Research, and the High-Energy Physics (HEP) community have established an International Data Grid project **[139]** with intent to apply the work to other scientific communities such as Earth Observation and Bioinformatics. The project objectives are to establish a research network for data Grid technology development, demonstrate data Grid effectiveness through the large-scale real world deployment of end-to-end application experiments, and to demonstrate the ability to use low-cost commodity components to build, connect, and manage large general-purpose, data intensive computer clusters.

The Data Grid project focuses on the development of middleware services in order to enable a distributed analysis of physics data. The core middleware system is the Globus toolkit with extensions for data Grids. Data in the order of several Petabytes will be distributed in a hierarchical fashion to multiple sites worldwide. Global namespaces are required to handle the creation, access, distribution, and replication of data items. Special workload distribution facilities will balance analysis of jobs in the Grid to maximize the throughput from several hundred physicists. Application and user access monitoring will be used to optimize data distribution.

The Data Grid project has a hierarchical machine organization with less data stored at lower levels of the hierarchy. CERN, which is Tier 0, stores almost all relevant data with several Tier 1 regional centers in Italy, France, UK, USA, and Japan supporting smaller amounts of data. It has an extensible schema based resource model with a hierarchical namespace organization. It does not offer any QoS and the resource information store is expected to be based on an LDAP network directory. Resource dissemination is batched and periodically pushed to other parts of the Grid. Resource discovery in the Data Grid is decentralized and query based. The scheduler uses a hierarchical organization with an extensible scheduling policy.

2.5.4 Globus: A Toolkit for Grid Computing

Globus [49] provides a software infrastructure that enables applications to view distributed heterogeneous computing resources as a single virtual machine. The Globus project is an American multi-institutional research effort that seeks to enable the construction of computational Grids. Currently the Globus researchers are working together with the High-Energy Physics and the Climate Modeling community to build a data Grid [1]. A central element of the Globus system is the Globus Toolkit, which defines the basic

services and capabilities required for constructing computational Grids. The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, data management, resource reservation, and communications. The toolkit provides a bag of services from which developers of specific tools or applications can select, to meet their own particular needs. Globus is constructed as a layered architecture in which higher-level services can be developed using the lower level core services [63]. Its emphasis is on the hierarchical integration of Grid components and their services. This feature encourages the usage of one or more lower level services in developing higher-level services.

Resource and status information is provided via an LDAP-based network directory called Metacomputing Directory Services (MDS) [122]. MDS consists of two components, Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS). GRIS implements a uniform interface for querying resource providers on a Grid for their current configuration, capabilities, and status. GIIS pulls the information from multiple GRIS services and integrates it into a single coherent resource information database. The resource information providers use a push protocol to update GRIS.

Thus MDS follows both push and pull protocols for resource dissemination. Higher-level tools such as resource brokers can perform resource discovery by querying MDS using LDAP protocols. The MDS namespace is organized hierarchically in the form of a tree structure. Globus offers QoS in the form of resource reservation. Globus provides scheduling components as part of its toolkit approach but does not supply scheduling policies relying instead on higher-level schedulers. Globus services have been used in developing many global schedulers, including Nimrod-G, AppLeS, and Condor/G.

2.5.5 Javelin

Javelin [82] is a Java based infrastructure for internet-wide parallel computing. The three key components of Javelin system are the clients or applications, hosts, and brokers. A client is a process seeking computing resources, a host is a process offering computing resources, and a broker is a process that coordinates the allocation of computing resources. Javelin supports piecework and branch-and-bound models of computation. In the piecework model, adaptively parallel computations are decomposed into a set of sub-computations. The sub-computations are each autonomous in terms of communication, apart from scheduling work and communicating results. This model is suitable for parameter sweep or master-worker applications such as ray tracing and Monte Carlo simulations. The latest Javelin system, Javelin 2.0, supports branch-and-bound computations. It achieves scalability and fault-tolerance by integrating distributed deterministic work stealing with a distributed deterministic eager scheduler. An additional fault-tolerance mechanism is implemented for replacing hosts that have failed or retreated.

The Javelin system can be considered a computational Grid for high-throughput computing. It has a hierarchical machine organization where each broker manages a tree of hosts. Resources are simple fixed objects with a tree namespace organization. The resources are simply the hosts that are attached to a broker.

Any host that wants to be part of Javelin contacts JavelinBNS system, a Javelin information backbone that maintains the list of available brokers. The host then communicates with brokers, chooses a suitable broker, and then becomes part of the broker-managed resources. Thus the information store is a network directory implemented by JavelinBNS. Hosts and brokers update each other as a result of scheduling work. Thus, Javelin uses demand resource dissemination. The broker manages the host-tree or resource information through a heap-like data structure. Resource discovery uses the decentralized query based approach since queries are handled by the distributed set of brokers.

Javelin follows a decentralized approach in scheduling, using work stealing and a fixed application oriented scheduling policy. Whenever a host completes an assigned job, it requests work from peers and thus load balancing is achieved.

2.5.6 Legion: A Grid Operating System

Legion [121] is an object-based metasystem or Grid operating system developed at the University of Virginia. Legion provides the software infrastructure so that a system of heterogeneous, geographically distributed, high performance machines can seamlessly interact. Legion provides application users with a single, coherent, virtual machine. The Legion system is organized into classes and metaclasses.

Legion objects represent all components of the Grid. Legion objects are defined and managed by their class object or metaclass. Class objects create new instances, schedule them for execution, activate or

deactivate the object, and provide state information to client objects. Each object is an active process that responds to method invocations from other objects within the system. Objects can be deactivated and saved to persistent storage. An object is reactivated automatically when another object wants to communicate with them. Legion defines an API for object interaction, but does not specify the programming language or communication protocol.

Although Legion appears as a complete vertically integrated system, its architecture follows the hierarchical model. It uses an object based information store organization through the Collection objects. Collections periodically pull resource state information from host objects. Host objects track load and users can call the individual host directly to get the resource information. Information about multiple objects is aggregated into Collection objects. Users or system administrators can organize collections into suitable arrangements. Currently, there is a global collection named "/etc/Collection" for the system that tracks HostObjects and VaultObjects which embody the notion of persistent storage. The users or their agents can obtain information about resources by issuing queries to a Collection.

All Classes in Legion are organized hierarchically with LegionClass at the top and the host and vault classes at the bottom. It supports a mechanism to control the load on hosts. It provides resource reservation capability and the ability for application level schedulers to perform periodic or batch scheduling. Legion resource management architecture is hierarchical with decentralized scheduling policies. Legion supplies default system oriented scheduling policies, but it allows policy extensibility through resource brokers. That is, application level schedulers such as Nimrod-G **[105]** and AppLeS **[44]** can change Legion default scheduling policies with user-centric policies.

2.5.7 MOL: Metacomputing Online Kernel

The MOL initiative is developing technologies that aim at utilizing multiple WAN-connected high performance systems as a computational resource for solving large-scale problems that are intractable on a single supercomputer. One of the key components of MOL toolbox is the MOL-Kernel [58]. It offers basic generic infrastructure and core services for robust resource management that can be used to construct higher-level services, tools and applications. The MOL-Kernel manages the resources of an institution's computing centers, provides a dynamic infrastructure for interconnecting these institutions, manages network faults, and provides access points for users.

The MOL-Kernel follows a three-tier architecture consisting of resource abstraction, management, and access layers containing resource modules, center management modules (CMM), and access modules respectively. The resource modules encapsulate metacomputing resources such as computing devices, scientific devices, applications and databases. All resource modules in a center are coordinated by CMM. This module is responsible for keeping its resources in a consistent state and makes them accessible outside of an institution. It acts as a gatekeeper and controls the flow of data between the center resources and external networks.

There is usually one CMM per institution, but it is possible to have multiple CMM in the case of large organizations. Failure of MOL-Kernel components results in only one institute becoming inaccessible. As long as a single CMM is available, the MOL-kernel remains operational. That means, organizations can leave or enter the metacomputing environment as they wish. The MOL-Kernel dynamically reconfigures itself to include or exclude the corresponding resources. In the MOL-kernel, CMM consistency is achieved by using a transaction-oriented protocol on top of virtual shared memory objects associated with each CMM. In order to make the global state available at all entry points, mirrored instances of shared active objects are maintained at each CMM. Whenever the state of a shared object changes, the new information is automatically distributed to the corresponding mirror instances. Extension of the MOL-Kernel is provided via typed messages and event handlers. Events are generated by user interaction with an access module or resource state changes. Messages are routed to either predefined or dynamically loaded custom event handlers.

The MOL follows a service Grid model with hierarchical cell-based machine organization. It adopts the schema based resource model and hierarchical name space organization. The global state is maintained in shared objects of each CMM (i.e., object based resource information storage). The resources and services themselves announce their initial presence to MOL (push protocol in information dissemination). The access modules/schedulers perform resource discovery and scheduling by querying shared objects. Although the resource model is schema based, its primary mode is service based. For example, if users

request an application (e.g. CFD-simulation) with a certain quality of service. MOL then finds those computers, which have this application installed, and asks them "which of you are powerful enough to provide the requested quality of service?" (i.e., decentralized scheduler). It then selects one or more to execute the request.

2.5.8 NetSolve: A Network Enabled Computational Kernel

Netsolve **[41]** is a client-agent-server paradigm based network enabled application server. It is designed to solve computational science problems in a distributed environment. The Netsolve system integrates network resources and provides a desktop application interface. The intent of Netsolve is to hide parallel processing complexity from user applications and deliver parallel processing power to desktop users. Netsolve clients can be written in C, Fortran, Matlab, or use Web pages to interact with the server. A Netsolve server can use any scientific package to provide its computational software. All component communications use TCP/IP. Netsolve provides resource discovery, fault tolerance, and load balancing.

The Netsolve system follows the service Grid model with hierarchical cell-based machine organization. The Netsolve-agents act as an information repository and maintain the record of resources available in the network. As a new node comes up, information such as its location and its services are sent to the Netsolve agent. Thus, the Netsolve Agent uses push resource dissemination. The Netsolve agent also acts as a resource broker and performs resource discovery and scheduling. The user requests are passed to an agent that identifies the best resource, initiates computations on that resource, and returns the results. Agents may request the assistance of other Agents in identifying the best resources and scheduling. Thus Netsolve has decentralized scheduler organization.

2.5.9 Ninf: A Network Enabled Server

Ninf is a client server based network infrastructure for global computing **[45]** similar to NetSolve. It allows access to multiple remote compute and database servers. Ninf clients access remote computational resources from languages such as C and Fortran using the Ninf client library. The Ninf client library calls can be synchronous or asynchronous in nature. The key components of the Ninf system are the Ninf client library, the Ninf metaserver, and the Ninf remote libraries. Ninf applications invoke Ninf library functions that generate requests. Requests are sent to the Ninf metaserver that maintains the information of Ninf servers in the network using an LDAP directory. The Ninf metaserver allocates resources on the appropriate servers for load balancing or scheduling. Ninf computational resources register details of available library services with the Ninf metaserver thus using a push protocol for resource dissemination. Ninf follows a flat model in machine organization, schema for resource model, and relational name space organization. The Ninf metaserver performs resource brokering, but the actual scheduling is done using extensible policies.

2.5.10 PUNCH: The Purdue University Network Computing Hubs

PUNCH [85][86] is a middleware testbed that provides operating system services in a network-based computing environment. The PUNCH infrastructure allows seamless management of applications, data, and machines distributed across wide-area networks. Users can run applications via standard Web browsers without requiring application changes.

PUNCH employs a hierarchically distributed architecture with several layers. A computing portal services layer provides Web-based access to a distributed, network-computing environment. This layer primarily deals with content management and user-interface issues. A network OS layer provides distributed process management and data browsing services. An application middleware layer allows the infrastructure to interoperate with other application-level support systems such as PVM [137] and MPI [138]. A virtual file system layer consists of services that provide local access to distributed data in an application-transparent manner. Finally, an OS middleware layer interfaces with local OS services available on individual machines or clusters of machines. The layers interoperate with a distributed resource management system and a predictive performance modeling sub-system in order to make intelligent resource allocation decisions.

2.5.11 Nimrod-G Grid Resource Broker

Nimrod-G [100][105] is a Grid resource broker that allows managing and steering task farming

applications on computational Grids. It uses an economic model for resource management and scheduling. Users formulate parameter studies using a declarative parametric modeling language or GUI with the experiment being run on the Grid. Nimrod-G provides resource discovery, resource trading, scheduling, resource staging on Grid nodes, result gathering, and final presentation to the user. Nimrod-G uses GRACE services to dynamically trade with resource owner agents to select appropriate resources. GRACE enabled Nimrod-G has been used for scheduling parameter sweep application jobs on the WWG testbed resources [98].

Nimrod-G follows the hierarchical and computational market model in resource management [105]. It uses the services of Grid middleware systems such as Globus and Legion for resource discovery and uses either a network directory or object model based data organization. It supports resource reservation and QoS through the computational economy services of the GRACE infrastructure. The users specify QoS requirements such as the deadline, budget, and preferred optimisation strategy. The Grid resource capability estimation is performed through heuristics and historical load profiling. Scheduling policy is application oriented and is driven by user defined requirements such as deadline and budget limitations. The load balancing is performed through periodic rescheduling.

2.6 Summary and Comments

There are currently a large number of projects and diverse range of new and emerging Grid developmental approaches being pursued. These systems range from Grid frameworks to application testbeds, and from collaborative environments to batch submission mechanisms.

There are many approaches and models **[108]** for developing Grid resource management systems. The systems we surveyed have for the most part focused on either a computational Grid or a service Grid. The only data Grid project that we have surveyed is the CERN Data Grid, which is in the initial stages of development. The other category of system is the Grid scheduler such AppLeS that is integrated with another Grid RMS such as Globus or Legion. These combinations are then used to create application oriented computational Grids with a certain degree of QoS. However, it can be observed that the existing Grid systems follow system centric approach to resource management. Among the various *Grid scheduling* systems, it can be observed that the Nimrod-G broker developed in this thesis is the only system that supports resource allocation and application scheduling algorithms driven by users' quality of service requirements.