2

Literature Survey

2.1 Introduction

Grid Computing is the next generation of IT infrastructure. It consists of collection of heterogeneous resources spread across multiple administrative domains. Grid Computing provides scalable, secure and extremely high performance mechanism for discovery and access to remote computing resources, in a seamless manner [35]. The reference [36] presents a middleware infrastructure, called as Internet Operating System (IOS), which aims at freeing application developers from dealing with non-functional concerns while seeking to optimize application performance and global resource utilization and [37] discusses about the adaptive middleware and programming technology for dynamic grid environments. Computational Grid has the potential to become the main execution platform for high performance and distributed applications. However, such systems are extremely complex. The possibility to discover a matching set of resources from a pool of an enormous amount of resources and mapping the jobs to the resources for execution is one of the challenging problems that a grid resource management system has to tackle. This chapter provides the survey of existing grid resource management systems and also an overview of existing resource discovery algorithms and scheduling heuristics.

2.2 Grid Resource Management System (GRMS)

Grid computing basically aims to harness the computing power of idle resources by borrowing the resources from the resource providers, rather than to buy more computational resources. Resource management in the computational grids deal with identifying requirements, matching resources to the applications, allocation of resources and scheduling and monitoring of the grid in a timely manner. Since a World Wide Grid (WWG) can consist of all the available resources connected over the internet, so finding a set of matching resources as per the requirements of the jobs/ tasks to be executed, and then generating and efficient job-resource mapping becomes very complex and tedious. Security and fault tolerance further add the complexity to the management of resources, and make this system more complex and challenging.

Traditional resource management systems can implement effective mechanisms and policies for the efficient use of the resources, as they have complete control over the resource. On the other hand in the grid resource management systems have resources distributed over different administrative domains. Hence the grid resource management system has to take care of resource heterogeneity, difference in usage, scheduling policies and security mechanisms. Co-allocation of resources (allocating resources in different sites to an application simultaneously) is also made complex due to site autonomy, which further adds the complexity to the resource management systems for computational grids [8]-[11].

A high level view of grid system is shown in Figure 1.4 (Chapter 1). In a grid system, an end user submits to the management system the job to be executed along with some constraints like job execution deadline, the maximum cost of execution. The function of the resource management is to take the job specification and from it estimate the resource requirements like the number of processors required, the execution time, and memory required. After estimating the resource requirements RMS is responsible for discovering available resources [11] and selecting appropriate resources for job execution. Finally schedule the jobs on these resources by interacting with the local resource management system.

2.2.1 Grid Resource Management System Taxonomy

Taxonomy for distributed system scheduling for static and dynamic task scheduling is presented in [38] and [39] respectively. Grid Resource Management System taxonomy based [40] on the resource management system architecture is given below. It is divided into following types: machine organization, resource namespace organization, resource discovery and dissemination and scheduler organization.

2.2.1.1 Machine Organization

The organization of the machines in a grid determines the scalability of the architecture. Machine organization also affects the communication patterns of the resource management system. Machine organization can be categorized as *centralized* or *de-centralized.* In the *centralized* organization there is a single controller or a designated set of controllers that perform the scheduling for all the machines. On the other hand in the *de-centralized* organization, the control is distributed among the machines in the grid. Centralized organization is generally suffered from scalability issues and hence not recommended for a grid. Other type of categorization can be done as a *flat*, *hierarchical* or *cell structure* organization. A *flat* organization is a kind of peer to peer organization where all the machines communicate with each other without any intermediate machine. In a *hierarchical* organization all the machine in the same level can communicate with the machines directly above or below or peer to them in the hierarchy. This organization is generally recommended for a grid as this is scalable. Last in this categorization is the *cell structure*, where are the machines in the cell communicate like flat organization but the communication with the machines outside the cell or in other cells is via some designated contact nodes only. Internal structure of the cell is generally hidden from other cells. Cells can be further organized as flat or hierarchical structure.

2.2.1.2 Resource Namespace Organization

The grid resource management system needs to maintain a globally named pool of resources. Resource namespace organization affects the other resource management functions like resource discovery and resource dissemination. Resource namespace can be organized as *relational*, *hierarchical* or *graph based.* A *relational* namespace divides the resources in to relations and used the concepts from relational databases to indicate the relationships between tuples in different relations. A *hierarchical* namespace follows a system of systems approach and a name is constructed by traversing down the hierarchy. A *hybrid* approach using *relational and hierarchical* namespace organization is also popular. Globus MDS is an example of hybrid namespace organization. Third approach is the *graph based* organization, where the resource are linked together and the resource names are constructed by following the links from one object to another.

2.2.1.3 Resource Discovery and Dissemination

Resource discovery services are an important function of resource management in a grid system. These are used by the scheduling system to obtain the information about the resources. Discovery process is initiated by network applications to find the suitable resources in a grid. Resource discovery approaches can be classified as *query based* or *agent based*. *Query based* systems enquire a centralized or distributed information store for the resource availability. In *agent based* systems information is gathered by discovery agents. The approaches for updating the resource information in the database constitute the resource dissemination process. This process is complimentary to resource discovery and involves advertising the resources in the grid. Dissemination protocol effect the amount of data transferred between systems. Resource dissemination can be either *periodic* or *on-demand*. In the *periodic* dissemination models the information is updated periodically. These can further categorized as *pull* or *push* model based on whether the information is updated / pushed by resource or the information store collects the information, respectively. *On-demand* resource dissemination models update information on the triggering of and event or some status change.

2.2.1.4 Scheduler Organization

Scheduling models can be categorized as *centralized, hierarchical* or *de-centralized*. Scheduler organization determines the structure of resource management system and its scalability. In *centralized* scheduler organization there is one scheduling controller and all the jobs are submitted to this controller. This single scheduler is responsible for scheduling all the jobs on to the resources. Due to the availability of information at one single location, optimal scheduling decisions are generally achieved, but this approach lacks in the scalability and fault tolerance. In a *hierarchical* organization scheduling controllers are organized in a hierarchy. This approach provides the advantages of scalability and fault tolerance over the centralized scheduler. Other approach is *de-centralized* model in which there is no centralized scheduler and scheduling is done by resource requesters and resource owners independently. This approach is scalable and maintains the site autonomy. Hence this is suitable for grid. Individual schedulers cooperate with each other in making scheduling decisions. This can be further categorized as *cooperative* and *non-cooperative* schedulers based on whether the individual schedulers cooperate with each other or not.

2.3 Grid Resource Management Systems Survey

Grid systems can be broadly categorized as compute, data or service based grids. This section discusses the details of some existing computational grid systems. Globus, Condor, Condor-G, Legion, Nimrod/G, Alchemi, GridWay and Gridbus Broker are studied.

2.3.1 Globus

Globus software [43]-[46] is designed to enable applications that federate distributed resources, whether computers, storage, data, services, networks, or sensors. Initially, work on Globus was motivated by the demands of “virtual organizations” in science. The collection of tools in the Globus toolkit provide basic services and capabilities like Resource Management, Resource Discovery, Security, Information Services etc. that are required for Grid Computing. Globus resource management system is a collaboration of different components which consists of *resource brokers*, *resource co-allocators* [45] and *resource manager* (GRAM) [46]. Globus is developed as a layered architecture in which the higher level services can be developed using the lower-level core services [47]. Globus emphasizes on hierarchical integration of grid services and components.

Globus uses a de-centralized scheduling model which supports QoS via resource reservation. Scheduling policies can be extended in the Globus is by the application level scheduler / brokers like Nimrod/G. Extensible Resource Specification Language (RSL) is used to specify the resource requests. Brokers are responsible for taking high level RSL specification and transforming them into more concrete specifications. Globus provides a component to implement resource management, data management, and information services, as shown in Figure 2.1.

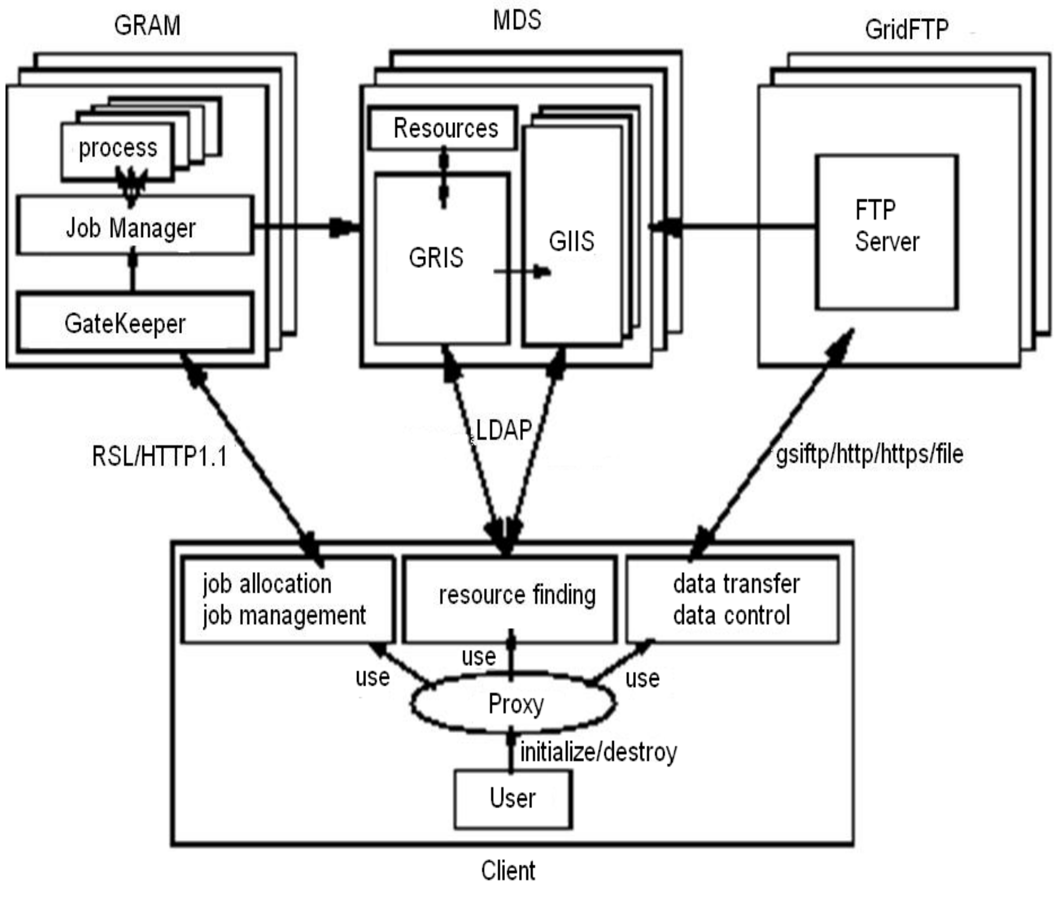

Figure 2.1: The system overview of Globus ToolKit

Resource discovery in the Globus is done by querying the LDAP (Lightweight Directory Access Protocol) based information store called as Metacomputing Directory Services (MDS). MDS consists of two components: Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS). GRIS is responsible for providing all resource discovery services. Resource information is updated by push dissemination and GIIS provides the global view of resources. Globus has a hierarchical namespace organization. Globus toolkit also provides an open source set of services and software libraries that supports the development of Grid systems and applications [48].

2.3.2 Condor

Condor [49] [50] is a high throughput resource management system that manages a heterogeneous pool of resources. It harnesses the computing power of idle resources by steeling the idle CPU cycles. Condor system follows a layered architecture. A set of resources managed by condor is known as condor pool. Condor keeps all the jobs submitted by the user in a queue. These jobs are then scheduled by the condor on to the machines in the pool transparent to the user [51][52]. Condor can also migrate a running job from one machine to another until it is completed [53][54].

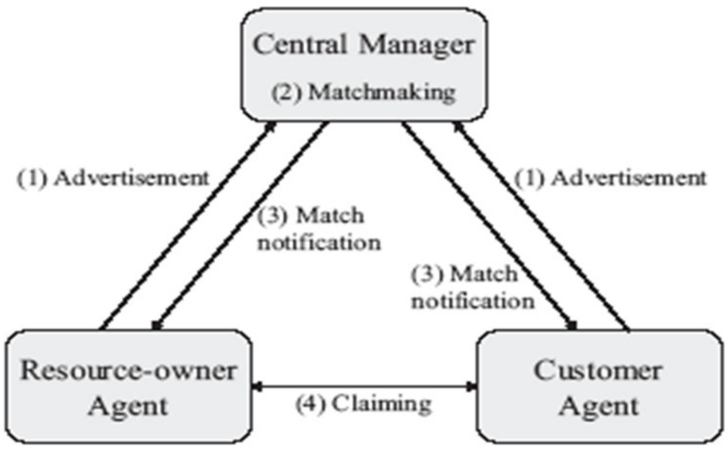

Figure 2.2: Condor Architecture

Condor uses a resource specification language, known as *Classified Ads* to specify the resource requests. ClassAds uses a semi structured data model and a query language as a part of data model that enables the advertising agents to include constraints to resource requests and offers and hence specify their compatibility.

Condor has a centralized scheduling model and uses a dedicated machine, known as Central Manager, which is responsible for scheduling the jobs onto the resources in the condor pool. Condor also supports pre-emption of jobs. In case a resource is withdrawn then already running jobs are check-pointed and pre-empted to other resources, thus ensuring the resource owner autonomy as well as proper completion of the jobs.

Condor can have multiple condor pools and each pool follow a flat RMS organization. Resource dissemination is through periodic push mechanism where each machine updates the resource status with the central information store owned managed by *Condor Collector*. A Condor resource agent runs on each machine periodically advertising its services to the collector. Collector is queried by condor matchmaker for resource discovery that is used to determine the matching resources and job requests [55], [56].

2.3.3 Condor-G

Condor-G [57] is the marriage of technologies from the Condor project and the Globus project. Condor-G combines the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource and job management methods of Condor to allow the user to harness multi-domain resources as if they all belong to one personal domain. So, it leverages recent advances in two distinct areas:

- ✓ security and resource access in multi-domain environments, as supported within the Globus Toolkit, and

- ✓ management of computation and harnessing of resources within a single administrative domain, embodied within the Condor system.

Condor-G provides the grid computing community with a powerful, full-featured task broker. Condor-G is mainly focused on job management part of Condor that lets the user to submit jobs into a queue, have a log detailing the life cycle of submitted jobs, manage input and output files, along with everything else that is expected from a job queuing system. Condor-G uses the Globus Toolkit to start the job on the remote machine instead of using condor-developed protocol. Hence Condor-G provides a "window to the Grid" for users to both access resources and manage jobs running on remote resources.

2.3.4 Nimrod/G

Nimrod/G is a grid resource broker [58] which is used as scheduling component in a new framework called Grid Architecture for Computational Economy (GRACE) [26] which is based on using economic theories for grid resource management systems. This is designed to run parametric application on computational grids [59]-[61]. It uses the middleware services offered by Globus toolkit. The main components of the GRACE infrastructure are a Trade Manager (TS), trading protocols and Trade Server (TS). TM is the GRACE client in the Nimrod-G resource broker that uses the trading protocols to interact with trade servers and negotiate for access to resources at low cost. Trade Server is the resource owner agent that negotiates with resource users and sells access to resources. TS uses pricing algorithms as defined by the resource owner that may be driven by the demand and supply. It also interacts with the accounting system for recording resource usage.

Resource discovery is accomplished by using the MDS services. Nimrod/G uses GRAM APIs to dispatch jobs over grid resources. Users can specify the deadline and budget for their jobs. Also resource providers can specify the price for offering the resource and other services. Nimrod/G tries to find the cheapest resources which can meet the deadline conditions for the jobs [60]. Though, Nimrod/G uses the middleware services provided by Globus Toolkit, but it can be extended to other middleware services also.

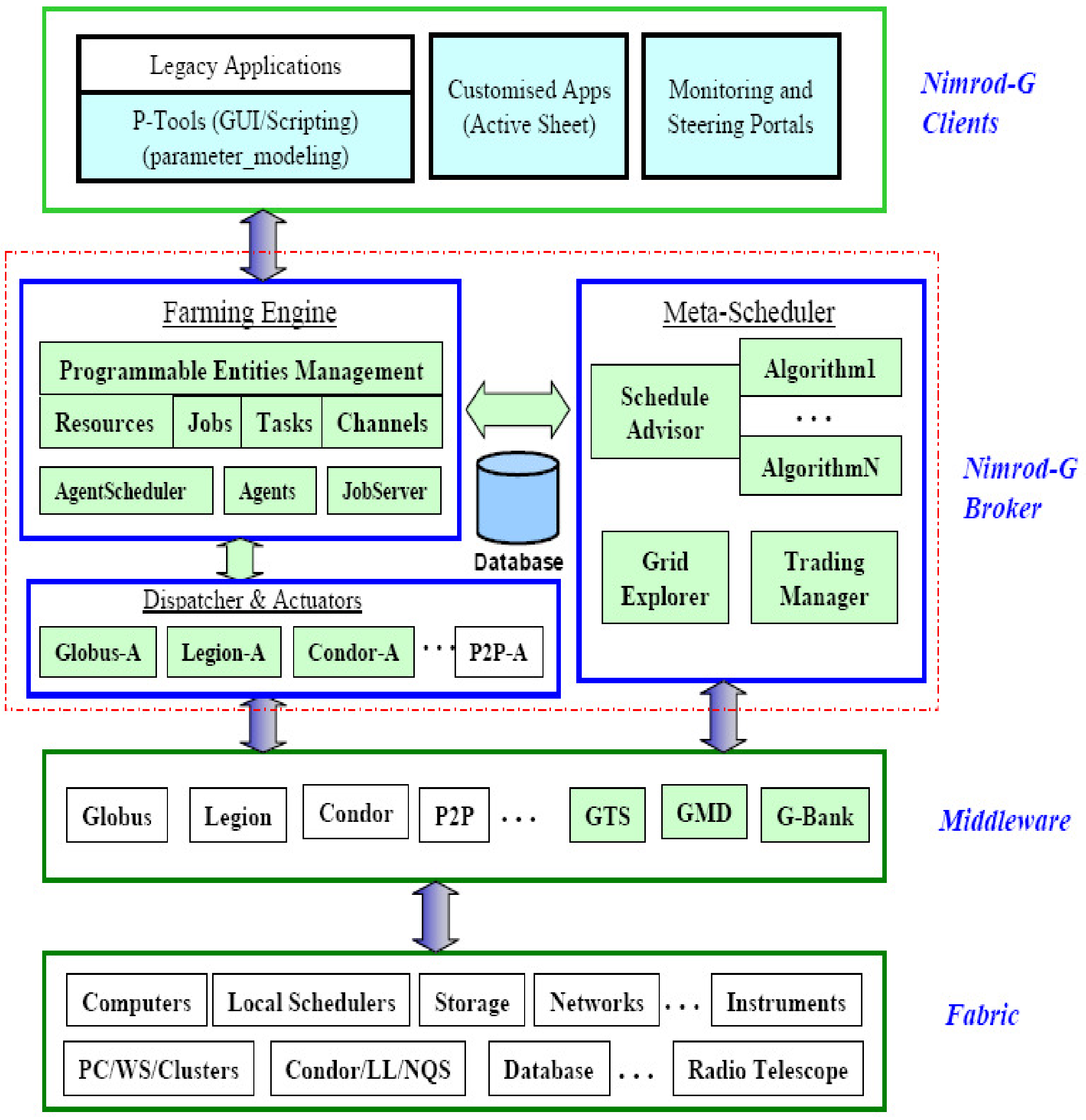

Figure 2.3: Layered Architecture of Nimrod/G [25]

Nimrod/G used a hierarchical machine organization, hierarchical namespace organization and market driven computational model for resource management.

2.3.5 Legion

Legion is an object based operating system for Grids [62]. This is a metasystem that offer software infrastructure for the Grids. This software infrastructure consists of a framework for scheduling different classes of applications with different type of scheduling strategies. The Legion objects are defined and managed by corresponding class. New instances for the classes are created and scheduled for execution and activated and deactivated [64][65].

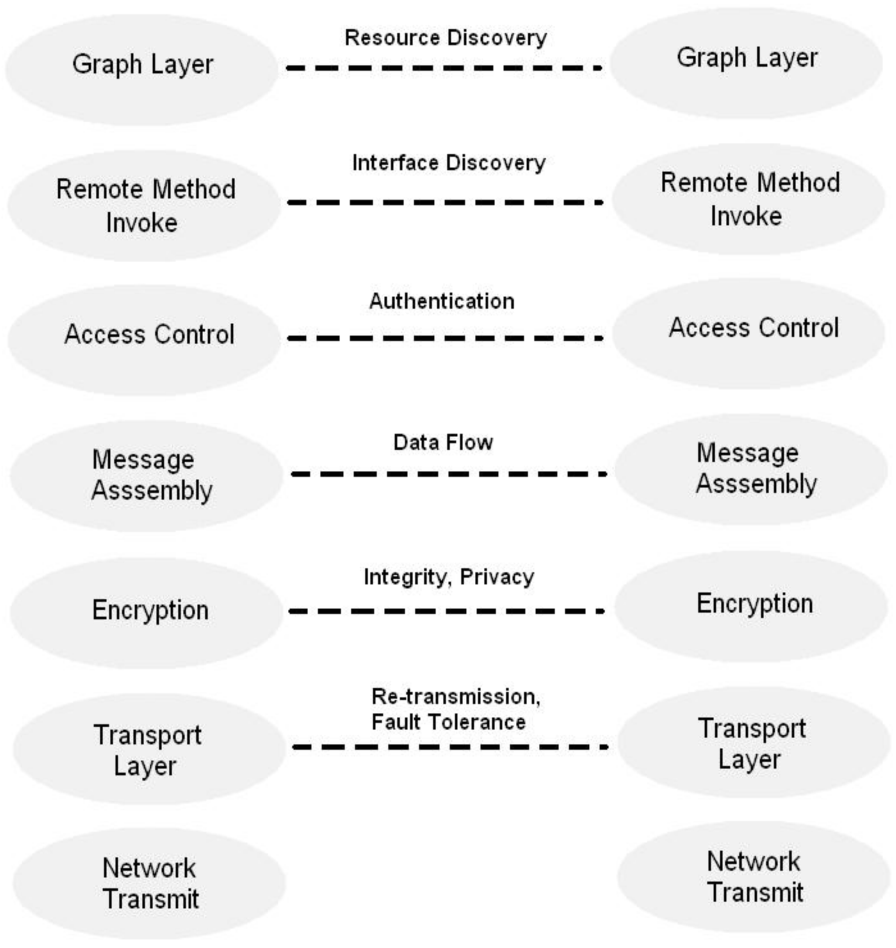

Figure 2.4: Communication in Legion objects through protocol stack

Legion has a hierarchical scheduling structure. Higher level scheduler does the job of global scheduling where it allocates the jobs to a cluster or a resource group. Actual scheduling of jobs on the resources with in the cluster is done by the local resource manager of that domain [66]. Legion objects are active processes and scheduling in the Legion is placing the objects on the processors. Co-allocation of resources in not supported.

Resources are organized in a graph based organization. Resource information is maintained in an object based information store and is managed through Collections [66]. Collection is a kind of information about multiple resources that is aggregated into single object. Periodic pull mechanism is used for resource dissemination where the Collections pull state information from host objects. Use of Collections makes the system more scalable where more collections can be added as required and these collections can communicate with each other to send and receive data from each other. Communication between any two objects goes through Legion protocol stack as shown in Figure 2.4.

2.3.6 Alchemi

Alchemi is .NET-based enterprise Grid computing and runtime machinery for creating a high-throughput resource-sharing environment [67]. The Alchemi Grid Computing framework was conceived with the aim of making Grid construction and development of Grid software as easy as possible without sacrificing flexibility, scalability, reliability and extensibility [68]. Alchemi architecture follows the master-worker parallel programming paradigm [69] in which a central component dispatches independent units of parallel execution to workers and manages them. Alchemi includes the runtime machinery (Windows executables) to construct computational Grid and a .NET API and tools to develop .NET Grid applications and Grid-enabled legacy applications. Reference [70] gives the performance comparison with Globus.

Alchemi consists of four major components: Manager, Executor, Owner and Cross-Platform Manager as shown in Figure 2.5

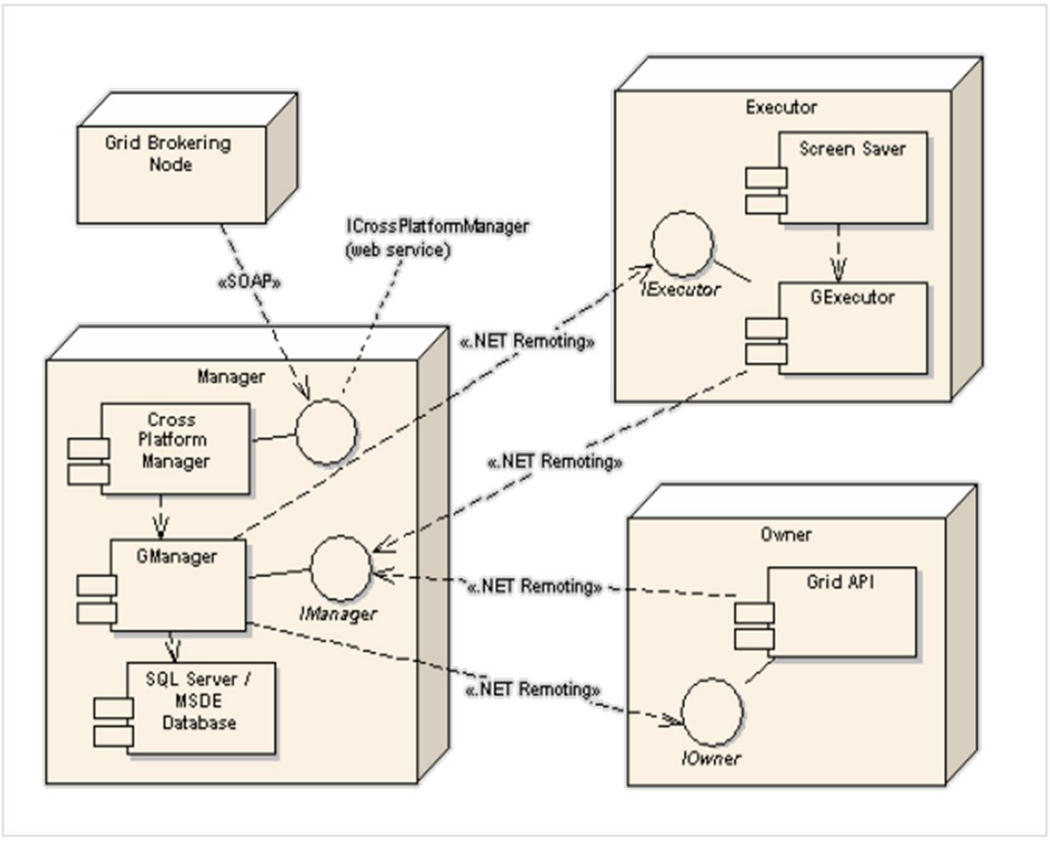

Figure 2.5: Alchemi architecture and communication between its components

Manager logically couples the Windows desktop machines running the instance of Alchemi Executor service. Executors register themselves with the Manager which in turn keeps track of their availability. Threads received from the owner are placed in a pool and scheduled to be executed on the various available executors. Owner provides an interface with respect to Grid applications between the application developer and the Grid. Cross-Platform Manager is an optional component of the Alchemi which provides a web services based interface to manage the execution of platform independent grid jobs.

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

2.3.7 GridWay

GridWay is a new experimental framework based on Globus that allows an easier and more efficient exuection of jobs on a dynmic Grid environment in a "submit and forget" fashion [71]. The core of the Gridway framework is a personal submission agent that performs all the scheduling stages and watches over the correct and efficient execution of jobs. Adaptation to changing conditions is achieved by dynamic rescheduling: once the job is initially allocated, it is rescheduled when a migration circumstance is detected.

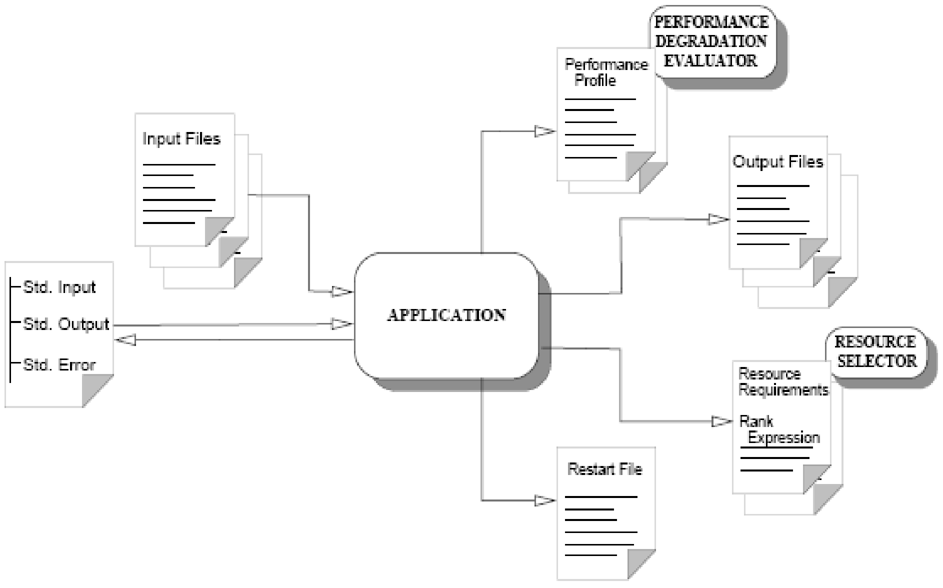

Figure 2.6: GridWay framework model for self adapting applications [71]

Job execution is performed in three stages by the prolog (prepares the remote system and stages the input files), wrapper (executes the actual job) and epilog (stages the output files and performs cleans up) modules which can be defined on a per job basis Migration is performed by combining the above stages. First, the wrapper is canceled and then the prolog is submitted to the new candidate resource and finally epilog is submitted to the old resource for clean up.

The submission system uses the resource selector (which evaluates the requirements for job scheduling) and performance evaluator module (which periodically evaluates the application performance) to detect performance slowdown and request re-scheduling. It also provides the application with fault tolerance capabilities. GridWay doesn't necessarily require source code changes in the application, but with minimal changes, applications could benefit from the self-adapting features provided by the framework.

2.3.8 Gridbus Resource Broker

The Gridbus broker extends the Nimrod-G computational Grid resource broker model to distributed data-oriented grids and extends Nimrod-G's parametric modeling language by supporting dynamic parameters [72]. Nimrod-G specializes in parameter-sweep computation with no functions for accessing remote data repositories and for optimizing on data transfer.

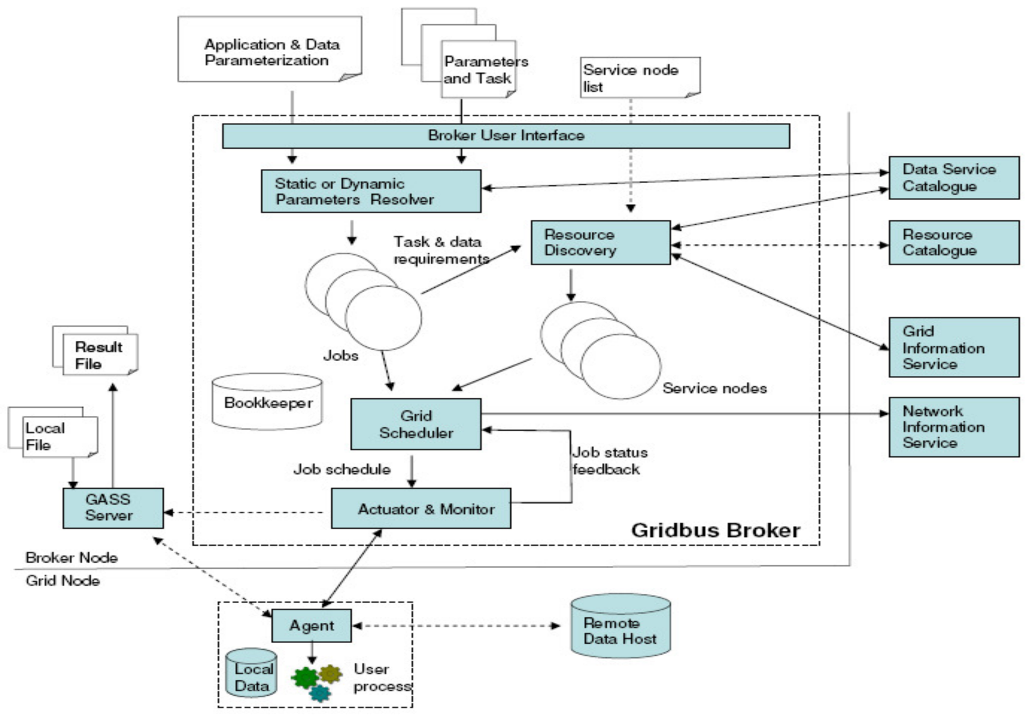

Figure 2.7: Gridbus Broker Architecture [72]

The architecture of the Gridbus broker is shown in Figure 2.7. The inputs to the broker are the tasks and the associated parameters with their values. A task is a sequence of commands that describe the user's requirements and the associated parameters can be static or dynamic. The task requirements drive the discovery of resources such as computational nodes and data resources. Task description is decomposed into jobs, which is sent to Grid node. The set of jobs along with the set of service nodes are an input to the scheduler. The scheduler matches the job requirements with the services and dispatches jobs to the remote node. For the jobs requiring remote data, it interacts with network monitoring service to optimize the data transfer. Actuator component dispatches the jobs to the remote nodes. 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.

2.4 Comparison of Grid Resource Management Systems

Table 2.1 below summarizes different grid systems discussed above, based on the taxonomy discussed above in section 2.2.1.

GRID SYSTEM	MACHINE ORGANIZA-TION	NAMESPACE ORGANIZA-TION	RESOURCE DISCOVERY AND DISSEMINA-TION	SCHEDULER ORGANIZA-TION
Globus	Hierarchical Cell machine organization	Hierarchical namespace	Distributed queries discovery, Periodic push dissemination	De-centralized scheduler. Use external schedulers for scheduling.
Condor	Flat machine organization	Hybrid namespace	Centralized queries discovery, Periodic push dissemination	Centralized scheduler
Condor-G	Hierarchical machine	Hybrid namespace	Distributed Queries	De-centralized scheduler.

	organization		discovery, Periodic Push dissemination	
Nimrod/G	Hierarchical Cell machine organization	Hierarchical namespace	Distributed queries discovery, Periodic dissemination	Hierarchical decentralized scheduler
Legion	Hierarchical machine organization	Graph namespace	Distributed Queries discovery, Periodic pull dissemination	Hierarchical scheduler
Alchemi	Centralized	Hierarchical namespace	Centralized queries discovery	Centralized scheduler uses First come first served Policy.
GridWay	Hierarchical Cell machine organization	Hierarchical namespace	Distributed queries discovery, Periodic push dissemination	De-centralized scheduler. Use external schedulers for scheduling.
Gridbus Resource Broker	Hierarchical Cell machine organization	Hierarchical namespace	Distributed queries discovery, Periodic dissemination	Hierarchical decentralized scheduler

Table 2.1: Comparison of different Grid Systems.

2.5 Grid Scheduling Heuristics

Scheduling in a Grid Resource Management System is mainly done by a scheduler or a broker. The scheduler at a lower level implements some scheduling heuristic to generate an optimal schedule. The reference [73] presents a cost-effective scheduling algorithm for message passing multiprocessor system where as duplication based scheduling algorithms are discussed in [74], [75]. Existing scheduling heuristics as given below are discussed and compared in [21].

2.5.1 Opportunistic Load Balancing (OLB)

OLB is very simple heuristic and assigns the jobs to resources, in an arbitrary manner as soon as the resource is available. Assignment is done without taking job's expected execution times into consideration. Main aim of OLB is to keep all machines as busy as possible. OLB generally results in very poor makespans.

2.5.2 Minimum Execution Time (MET)

MET works in contrast to OLB and assign each job, in an arbitrary manner to the resource with the best expected execution time for that job, regardless of the availability of the resource. MET focuses on assigning each job on a best resource for it. MET tries to find good job-resource pairings, but because it does not consider the current load on a resource, it will often cause load imbalance between the processors.

2.5.3 Minimum Completion Time (MCT)

MCT combines the benefits of both OLB and MET, by assigning each job to the resource with minimum expected completion time for that job. MCT tries to avoid the circumstances in which OLB and MET perform poorly.

2.5.4 Min-min

Min-min establishes the minimum completion time for every unscheduled job (in the same way as MCT), and then assigns the job with the least minimum completion time (hence Min-min) to the resource which offers it this time. Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped jobs during each mapping decision and MCT only considers one job at a time.

2.5.5 Max-min

Max-min is very similar to min-min. Again the minimum completion time for each job is established, but the job with the maximum minimum completion time is assigned to the corresponding resource. Max-min is based on the intuition that it is good to schedule larger jobs earlier on so they won't cumulate at the end causing a load imbalance.

2.5.6 Duplex

Duplex heuristic is a combination of both Min-min and Max-min heuristics. Duplex performs both Min-min and Max-min heuristics and the uses the better solution.

2.5.7 Genetic Algorithms (GAs)

GAs are an evolutionary techniques that is used to search for optimal solution in a very large search space. GAs are inspired from human genetics and generally work by encoding the problem in the form of chromosomes. GA operators like crossover and mutations are applied and new generations are evolved. Fitness is computed after every generation and further exploration is stopped as soon as an acceptable fitness value is achieved. GAs are till now best known heuristics for scheduling.

2.5.8 Simulated Annealing (SA)

SA is an iterative technique that considers only one possible solution (mapping) for each job at a time. This solution uses the same representation as the chromosome for the GA. SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space. This probability is based on a system temperature that decreases for every iteration. As the system temperature "*cools*" it is more difficult for poorer solutions to be accepted.

2.6 Grid Resource Discovery Algorithms

Number of algorithms has been developed for Resource Discovery process in Grids. Few of popular algorithms are discussed below.

2.6.1 Flooding Algorithm

This algorithm is based on agent-based approach and is widely used by Internet routers [76]. It is also used in peer to peer file sharing system, advertising certain routing tables or link states to routers and some part of routing protocols that are used in ad-hoc wireless networks. Every node acts as a transmitter and receiver and every node tries to send every message to every node of its neighbor. Newly added edge is not used for any communication. Direct communication exists only in between initially existing set of neighboring edges of the network. The required number of rounds of this algorithm is equivalent to the diameter of the graph. This algorithm can be very slow if not started with a graph, which has small diameter. This algorithm is not quite good for Grid Computing, due to the size of the Grid environment which, consists of many nodes.

2.6.2 Swamping Algorithm

Agent based approach acts as the base approach for implementing this algorithm. Swamping algorithm is identical to flooding algorithm except that this algorithm allows a node to connect not only with the set of initial neighbors but also with all of its current neighbors [76]. The main advantage of this algorithm is that it has better speed than the flooding algorithm and it needs $O(\log n)$ rounds to converge to a complete graph. The disadvantage of this algorithm is, its communication complexity grows very quickly.

2.6.3 Random Pointer Jump Algorithm

In this algorithm, in each round, each node contacts with a random neighbor, and then this random neighbor sends all of its neighbors to the sender node. Finally sender neighbor and random neighbor's neighbors get merged [76]. This algorithm claimed that a strongly connected graph with *n* nodes needs *O(n)* complexity time to converge to a complete graph it is good choice for strongly connected graphs. This can further be categorized as *Before Random Pointer Jump* or *After Random Pointer Jump* based on whether the connection to the neighbor is made before passing the information or after passing the information to the other node.

2.6.4 Name Dropper Algorithm

This is an algorithm for querying resources in a weakly connected network where it is assumed that all machines already know each other [76]. The algorithm, which takes $O(log_2 n)$ rounds to learn each other, $O(n^2 \ log_2 n)$ number of connections and $O(n^2 \ log_2 n)$ number of pointers in a network. It is claimed [76] that Name Dropper algorithm is more efficient in terms of required time and total number of network communications compared to four other algorithms, mainly *flooding algorithm, swamping algorithm, random pointer jump algorithm*, and *random pointer jump with back edge* algorithm.

2.6.5 Time-To-Live based Reservation Algorithm

The time-to-live based reservation algorithm is fully decentralized Resource Discovery algorithm and is most suitable for Grid Computing" [77]. This algorithm is based on a simply unicast request transmission that can be easily implemented. The addition of a reservation algorithm enables resource discovery mechanism to find more available matching resources. The deadline for resource discovery time is decided with time-to-live (TTL) value.

2.6.6 Discovering Intermittently Available Resources (DIAR) Algorithm

This algorithm is used to discover the occasionally available resources in multimedia environment [78]. Different policies for a QoS based Resource Discovery service for a given graph theoretic approach is defined. The performance evaluation of QoS policies based on different time-map strategies in a centralized system proves that randomized placement strategies and increased server storage can facilitate better performance to discover a particular resource.

2.6.7 Absorption (Randomized Resource Discovery) Algorithm

The Randomized Resource Discovery algorithm, also called “absorption algorithm”, is based on a strongly connected graph and is developed in two stages [79]. First stage is to determine the ultimate leader in the network where a network of *n* nodes is partitioned into clusters; each cluster has its leader node and all nodes in a cluster know their leader. In second stage, the ultimate leader of the network broadcasts the pointers to the each member of the network in one time step. The authors claimed that this absorption algorithm takes $O(\log n)$ steps time complexity, can send $O(n \log n)$ number of messages, and can pass $O(n^2 \log n)$ number of pointers with high probability.

2.7 Summary

This chapter provides the review of grid resource management systems and core algorithms. Taxonomy for assessing the Grid Resource Management System is given and a survey of existing grid middleware is presented to show how different components had been implemented in grid systems. Following the comparison of different Grid systems, a detailed review of existing grid scheduling and grid resource discovery algorithms is presented. Next chapters (Chapter 3 and Chapter 4) focus on the details of proposed meta-heuristics based algorithms for grid scheduling and proposed ACO guided mobile agent based algorithm for grid resource discovery.