Chapter 2

Literature Survey
Chapter 2: Literature Survey

2. Literature Survey

Compute grid enables resource sharing and aggregation providing high computing power at cost effective rate compared to that provided by centralized high end super computers or clusters. Resource sharing is made possible by layer in grid architecture called Grid Resource Management and Scheduling, which is the central and important part of any grid implementation.

Exhaustive literature study related to the research area has been undertaken to survey the related work done and investigate the issues and challenges in QoS oriented grid resource management and scheduling.

Various studies, survey and research papers about grid implementations, existing grid middleware and technologies for designing and deploying grid have been investigated in detail with their advantages and limitations. Existing grid resource management and scheduling models have been studied and their comparative study and analysis have been made. Based on the findings of literature study, the need and requirement analysis for QoS oriented grid model, QoS algorithm and mechanisms has been proposed.

The sections in literature survey and review are as below:

- Grid Middleware
- Distributed Resource Management System (DRMS)
- Grid Resource Management and Scheduling (GRMS)
- Study of Existing Models for Grid Resource Management and Scheduling
- Analysis and Comparative Study of Grid Resource Management and Scheduling Models
- Grid Scheduling Algorithms
- Quality of Service (QoS) in Grid Computing
- Agent Based Grid Computing
- Summary and Outcome of Literature Survey
2.1 Grid Middleware

Grid middleware is a layer of software enabling sharing, organization and integration of distributed grid resources. Grid middleware handles the different functions of grid system and exists between the grid application and diverse grid components like operating system, network and computer resources. While different grid applications in different domains may have specialized different requirements, but there are certain common requirements like resource discovery and job submission which are provided by grid middleware. It provides various fundamental generic grid services like authentication and security, job submission, remote data transfer, grid communication and grid information directory. Grid middleware publishes the various grid functionalities through high level API (Application Programming Interfaces) for development of grid schedulers and applications. It facilitates the resource sharing and negotiation between resource providers and grid applications and end users. Job execution services also allow users to interface with batch queuing systems. Examples of grid middleware are Globus toolkit [48], UNICORE (UNiform Interface to COmputing REsources) [54], Legion [55] and Gridbus.

Grid middleware provides basic grid services to higher layer of Grid application and portals in the grid architecture. As specified by Parvin Asadzadeh et al. in [43], grid middleware can be classified into different levels like core grid middleware and user level middleware.

*Grid Fabric Layer* consists of diverse remote distributed resources like computers, storage, networks to be aggregated by the grid system middleware.

*Core Grid middleware* offers basic grid services and functionality such as remote job submission, process management, resources allocation protocols, data storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading. These services abstract the complexity and heterogeneity of the fabric level by providing a consistent method for accessing distributed resources.
<table>
<thead>
<tr>
<th>Applications and Portals</th>
<th>Grid Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Scientific, Engineering, Life Science)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development Environment and Tools</th>
<th>User Level Grid Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>(languages, compilers, workflow generators)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grid Resource Brokers</th>
<th>Core Grid Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Resource Management and Scheduling)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fundamental grid services</th>
<th>Grid Fabric Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Security, Job submission, Resource Information, Data transfer and storage)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local Resource Managers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Operating Systems, Batch and Queuing systems,</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Networked Distributed Grid Resources</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(Computers, Storage, Network)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.1: Layers of Grid Middleware [43]

*User-level Grid middleware* provides user specific functionality to grid users of different domains and utilizes the interfaces provided by the low-level middleware to provide higher level abstractions and services. These include application development environments, programming tools and resource brokers for managing resources and scheduling application tasks for execution on global resources.

*Grid Applications* and grid web based interface portals use the various grid middleware functionality and services to design, develop and schedule the grid applications and grid jobs to remote grid resources and retrieve the results.
Chapter 2: Literature Survey

2.2 Grid Middleware Technologies

Grid middleware is the core part of the grid system providing fundamental grid services and uniform and seamless access to diverse remote resources in the grid. Over the years, several grid middleware including software toolkits and systems have been developed as a result of research projects undertaken by academic and scientific community. Some of the prominent grid middleware widely deployed are Globus Toolkit, UNICORE, Gridbus among others.

2.2.1 UNICORE

UNICORE (Uniform Interface to Computing Resources) is a part of European Middleware Initiative (EMI) which provides client and server grid software for sharing of distributed resources in a seamless manner in intranets and internet [54]. UNICORE is an open source standards based grid middleware providing end-to-end components at all levels of grid architecture.

UNICORE architecture consists of three layers - client layer, service layer and system layer.

UNICORE site (USite) is the collection of computers resources participating in the grid that client can connect to. A USite provides various atomic services like computing or data services. UNICORE gateway is the secure entry point to the site using security mechanisms to access the grid resources in the site.

Some of the key features of UNICORE are as below:

- Supports the grid standards by Open Grid Forum (OGF) and Advancing Open Standards for Information Society (OASIS) like Open Grid Services Architecture (OGSA), Web Service Resource Framework (WSRF).
• Different Clients like Command line, User friendly Graphical client supporting workflows for job submission serving the need of various application domains and scientific communities.
• Support for Multiple Operating and Batch systems: like Windows, MacOS, Linux and Unix Systems as well as batch systems like Torque, LSF, LoadLeveler etc.
• Security mechanisms using X.509 certificates based authentication, authorization and support for proxy certificates and creation of virtual organization (VO).
2.2.2 Globus Toolkit (GT)

Globus Toolkit (GT) is an open source grid middleware for creating grids. It provides various grid middleware services, protocols and implementation of grid standards for distributed resource sharing and access across different security and administrative domains. It enables sharing and aggregation of diverse resources like computing power, storage, databases and other resources across geographical and organizational boundaries. GT supports various Global Grid Forum (GGF) defined grid standards like OGSA (Open Grid Services Architecture), OGSI (Open Grid Service Infrastructure), WSRF (Web Services Resource Framework), Job Submission Description Language (JSDL). GT also implements various Open Grid Forum (OGF) defined protocols like Grid Resource Allocation & Management Protocol (GRAM), Monitoring and Discovery Service (MDS), Grid Security Infrastructure (GSI), Global Access to Secondary Storage (GASS) and GridFTP.

In this research work, GT version 4.2.1 is used. Various GT components address fundamental grid services and functionality for building grid infrastructure for executing grid applications. GT components have their library and the functionality is accessible using public APIs and WSDL (Web Service Description Language) interfaces. GT is modular and its various grid components can be installed and used independently in different nodes of the grid system. The different modules of GT are described below [57]:

The globus architecture is based on SOA concepts and provides various grid middleware services in form of SOAP web services which can be invoked by any grid application clients directly or by grid resource broker.

Grid Service Implementations

The various grid services are executed in the Globus container (Java WS Core) which are required to be used by grid applications are – execution management (GRAM), data access and movement (GridFTP, RFT), authentication and authorization services
Chapter 2: Literature Survey

(MyProxy, Delegation, SimpleCA) and discovery and monitoring of grid resources (WebMDS, Index, Trigger).

Containers

The Java, Python and C containers can be used to deploy user defined grid application web services. The containers provide the basic services like security, discovery for developing web services and also host the web services of fundamental grid services.

Client libraries

They provide the uniform abstraction and mechanism for invoking the inbuilt GT services and user defined services from grid client applications like Java, Python or C.

Globus Toolkit Components

The various fundamental components to create a grid infrastructure like common runtime, execution management, data management, security, information required are provided by globus toolkit.

Globus Architecture

Execution and Job Management components

This component provides the functionality needed for initiation, monitoring, coordination and control of remote job executions called “jobs” in grid system. This component performs remote resource allocation, job submission, data transfer associated with job submission, monitoring and control for executing grid applications. GT4 provides implementation of GRAM4 (web service based) and GRAM2 (non-web services legacy version) protocols which are standardized Grid Resource Allocation and Management protocol.
Figure 2.3: Globus Toolkit 4 Architecture, components and interaction [49]

Figure 2.4: Globus Toolkit components [49]
GRAM protocol provides uniform interface to requesting and using remote system resources managed by diverse local schedulers (like Fork, Condor, PBS, LSF) and operating systems for execution of grid jobs [58] [59].

**Common runtime**

Provide a set of fundamental libraries, tools and runtime environment to build and deploy grid applications and services (both WS and non-WS service based).

**Security**

Grid Security allows establishing the identity of users and grid applications in distributed environment (authentication) and authorization of users and grid applications. It also protects communication and allows managing public-key-infrastructure (PKI)-based user credentials. Using the Security components, based on the Grid Security Infrastructure (GSI), we can make sure that our communications are secure [60].

**Data management**

Grid data management allows location, transfer and managing large data sets required for executing grid applications. GT4 provides various tools for data management:

- Grid FTP - providing high-performance and reliable data transfer
- RFT - allows for managing parallel multiple data transfer
- RLS - manage location information for replicated files

**Information services**

Provides MDS (Monitoring and Discovery Services) for discovering, indexing and distributing resources and theirs status for resource management and scheduling grid applications on distributed resources.
2.2.3 Gridbus

Gridbus is an open source grid middleware developed under Gridbus project by CLOUDS lab, University of Melbourne. The project involves design and development of service-oriented grid middleware technologies to support eScience and eBusiness applications [61].

Features of the Gridbus middleware are:

- Utility oriented computing model supporting clusters and grids
- Uses various economic based scheduling models for effective utilization and commoditization of grid resources
- Trading of grid services based on demand and supply model

Gridbus toolkit contains different components providing different grid functionalities [63]:

Figure 2.5: Architecture of Gridbus Middleware [62]
Chapter 2: Literature Survey

Gridbus Broker – is a competitive economy based data grid service broker for scheduling data intensive applications across linux and window based grid resources

Alchemi - .NET based grid framework for creating desktop grids

GridSim – is a toolkit for modeling, simulation and performance evaluation of time and space shared grid resources

Grid Workflow Management System – provides gridbus workflow engine
Web service based Grid Market Directory – for trading of grid services

G-Monitor - Portal for remote monitoring and management of Grid applications executing on remote grid resources

Grid Development tools – visual grid application development tools for rapid development of distributed applications
Table 2.1: Comparison of Grid middleware

<table>
<thead>
<tr>
<th></th>
<th>UNICORE</th>
<th>Globus Toolkit</th>
<th>Gridbus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Vertically integrated grid software providing seamless access to intranet and internet grid resources</td>
<td>Open source toolkit providing core grid middleware</td>
<td>Open source Data grid broker and middleware for clusters and grids</td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td>Provides the client and server software for ready to run grid system</td>
<td>Provides core services, interfaces and protocols for building grid across institutional and geographic boundaries</td>
<td>Provides utility and economy oriented grid computing model Supports GGF-JSDL, Parameter-sweep execution model</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Three tier architecture User, server, and target system tier</td>
<td>N-tier distributed architecture</td>
<td>N-tier distributed architecture</td>
</tr>
<tr>
<td><strong>Job Submission</strong></td>
<td>Abstract Job Object (AJO) model</td>
<td>GRAM (Grid Resource Allocation and Management) Protocol</td>
<td>Supports GGF-JSDL, Parameter-sweep execution model</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>X.509 Certificates</td>
<td>Grid Security Infrastructure</td>
<td>Single sign-on through X.509V3 certificates</td>
</tr>
<tr>
<td><strong>Organization</strong></td>
<td>European Middleware Initiative</td>
<td>Globus Alliance</td>
<td>GRIDS lab, University of Melbourne, Australia</td>
</tr>
</tbody>
</table>
2.3 Distributed Resource Management System (DRMS)

Distributed Resource Management Systems (DRMS) manage and control the distributed heterogeneous resources in high performance parallel computation systems. The objective of DRMS is to maximize the utilization of distributed resources, system throughput, load balancing and satisfy the resource requirements of jobs submitted by end users. In the literature, DRMS are also commonly referred as batch systems, job management system, distributed scheduling or queuing systems. Popular examples of such systems - which are widely studied, deployed and used by distributed systems community are Portable Batch Systems (PBS) [64], Condor [65], Platform Load Sharing Facility (LSF) [66], Sun Grid Engine (SGE) [67] and IBM Load Lever [68].

The end users use DRMS to submit sequential or parallel jobs which require distributed resources like CPU, network bandwidth, primary and secondary memory managed by DRMS to execute jobs successfully. DRMS provide support for cluster and parallel computing libraries and systems to execute high performance computing applications. The study of DRMS, their requirements, features and comparison has been done in studies like [69] and [70].

Comparative study of various performance oriented features like response time, system throughput, system utilization and job execution efficiency of LSF, PBS, SGE and Condor has been done in [71].

DRMS have three main subsystems [72]:

**Job Management System (JMS)**

Provides interface between distributed resources and users by taking care of job submission, job control, queuing and job monitoring. It provides support for executing different types of distributed jobs like batch jobs, parallel jobs with MPI support, array jobs, interactive jobs and complex jobs with workflow and interdependencies. The
functionalities provided by JMS includes job submission with job specifications like job executables, arguments, input/output files, resource requirements and job execution environment, job priority, job status notification and job migration for fault tolerance.

**Physical Resource Management**

This subsystem provides interfacing and control of physical hardware resources like CPU, memory, secondary storage and network bandwidth. It also provides resource usage accounting, static and dynamic information of resources.

**Scheduling and Queuing system**

This subsystem helps in efficient execution of jobs and optimum utilization of distributed resources using scheduling policies and priority based queuing.
2.4 Grid Resource Management (GRM)

Grid Resource Management system is a type of distributed resource management system supporting grid systems. The term Grid Resource Management (GRM) specifies the set of functionalities and services that are needed for managing and utilizing the available pools of remote resources in a Grid environment and for providing means to access these resources [73][74].

Resource management refers to the operations used to control how capabilities provided by grid resources and services are made available to other entities, whether users, applications or services [75]. GRM is the process of identify application requirements, matchmaking the grid application requirements with available grid resources, perform scheduling decision of grid applications, remote resource allocation and monitoring of grid applications.

GRM acts as an interface bridge between remote grid resources and grid users and also provides various services to the end users for executing grid applications [76].

Resource managers like operating system and batch schedulers in clusters have complete information and control of all the resources managed by them and thus can implement the policies and mechanism to manage them. While Grid resource managers must abstract the heterogeneity among the grid resources with respect to type of resource, owners/providers, cost and performance, access policies, etc. and provide unified access to resources. Grid resources management system has to interact with different local resource managers for accessing the different resources. Global Grid Forum has defined service based resource management architecture by providing different Open Grid Service Architecture (OGSA) based services for invoking GRM functions. OGSA provides a general and generic model for a service oriented grid platform and it includes grid resource management functionalities [77].
Resources that need to be managed by GRM are [78]:

- Compute (nodes, processors etc.)
- Storage (space, location etc.)
- Data (availability, location etc.)
- Network (bandwidth, delay etc.)
- Software (components, licenses etc.)
- Services (functionality, ability etc.)

Grid resource management involves: Allocate a resource, authenticate a resource, authorize a resource, assurance of resource availability, accounting and auditing of a resources [8]. The functions of Grid Resource Management are as under [79]:

- Discovery of resources based on requirements of grid applications and fulfilling constraints
- Maintaining the information about grid resources and their attributes like amount of processing power, main memory, secondary storage, hardware and software environment.
- Static and Dynamic information about the status and attributes of grid resources.
- Resource allocation and co-allocation of resources [80]
- Status monitoring about the condition/status of submitted grid jobs and used resources. Take corrective action in case of resource failure.
- Job Scheduling is required so that the end users do not have to coordinate the access to different resources required to execute the grid application.
- Quality of Service and Resource reservation requirements.

The following issues make grid resource management and scheduling complex and crucial [81] [82]:

- Site autonomy
- Resource heterogeneity
Chapter 2: Literature Survey

- Difference in resource usage and scheduling policies,
- Security mechanisms, and
- requirement of interfacing and inter operation with local with local resource management systems
- Need for the negotiation between resource users and resource providers

Grid resource management problems and generalized mechanisms to address those problems based on Open Grid Services Architecture (OGSA) have been proposed in [74].

In [83], requirements of grid resource management systems, their abstract model and comprehensive taxonomy of grid resource management systems based on different criteria is presented:

Machine organization - determines how the participating nodes in the grid system are organized and communication structure between them.

Resource model - can be schema model where the resource is described using schema description language and has query mechanism to obtain the attributes and status of the grid resource. In object resource model, the operations on the resource are described as part of resource model abstraction.

Resource namespace organization – resource namespace is used to identify and discover the grid resources. Hierarchical namespace organizes grid resources into hierarchical structure based on physical network structure.

Resource discovery mechanism – provides the information about available grid resources and their current status for grid scheduler to take scheduling decisions. Resource discovery mechanism is classified into based on query or agents. Query based resource discovery uses network directory like Globus MDS [84] to execute parameterized queries. Agent based mechanism do not rely on query based approach and uses mobile agents that communications to remote resource agents to obtain the resource status.
QoS Support – has to be provided by all components of the grid and by network, storage and processing resources is required. Resource management systems should allow applications to specify QoS in application execution request. QoS support is classified into soft QoS and hard QoS. In Soft QoS, explicit QoS attributes for resource requests are specified but not monitored and enforced through strict policy. Hard QoS is enforced when all the components and resource guarantees and enforces the desired QoS levels requested by the grid clients.

**Resource Discovery**

Resource discovery is an important capability required in grid infrastructure which facilities grid resource management [85]. It is important to keep track of available distributed grid resources with their attributes and status for effective scheduling of grid applications. To achieve this, efficient mechanisms and protocols for grid resource management and discovery are needed.

Various authors have proposed the importance of trustworthiness and security in grid resource sharing and discovery [86] [87].

Resource discovery in a dynamical grid system based on re-routing tables has been proposed in [88]. Semantic based grid resource discovery has been proposed in [89] [90].

Semantic based component has been proposed in grid architecture to have semantic based description of grid resources based on ontology template. Semantic grid uses metadata and well defined meaning and description of grid information and services using semantic web concepts like Resource Description Framework (RDF). This can help in better interaction and automatic discovery of grid resources for forming virtual organizations (VO).
2.5 Grid Scheduling

Job scheduling is an integrated part of parallel and distributed computing. Intensive research has been done in field of scheduling for distributed systems. However, with the emergence of the computational grid, new scheduling approaches and algorithms are required for addressing new requirements and concerns arising in the grid environment.

“Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites [41].”

A scheduling problem is specified by a set of machines, a set of tasks, an optimality criterion, environmental specifications, and by other constraints [91].

To make effective use of distributed heterogeneous resources in grid computing, efficient scheduling algorithm is important. The goal of grid scheduling is optimal assigned of jobs to grid resources fulfilling user QoS constraints. It involves performing matchmaking and establishing mutual agreement between resource consumer and resource providers. Grid resource consumers are resource intensive grid applications that benefit by aggregating the grid resources provided by resource providers. Grid resource providers volunteer their idle resources for use by grid resource consumers to maximize their resource utilization.
Chapter 2: Literature Survey

The general concepts and terms used in Grid scheduling terminology are as under:

Task: The term is used to indicate the smallest unit of atomic work which can be scheduled by the grid scheduler

Grid Job (application): submitted by the grid user is the set of atomic tasks which needs to be executed in particular order

Resource: indicates the grid resources like processor, network bandwidth, secondary storage provided by machines (nodes) part of the grid system

Site (node): is independent grid entity comprised of multiple grid resources

Task scheduling: refers to the process of mapping/assigning tasks to different resources in the grid based on scheduling objectives or criteria

There are three main phases of scheduling on a grid [53]:

1. Phase one is resource discovery, which generates a list of potential resources.
2. Phase two involves gathering information about those resources and choosing the best set to match the application requirements.
3. In phase three the job is executed, which includes file staging and input output.

In the second phase, the choice of the best pairs of jobs and resources is a complex problem (NP-hard) to map existing tasks to accessible storage and computational resources to utilize them in the optimistic manner. In the literature, many heuristics have been proposed to solve the NP-complete problem of efficiently scheduling grid tasks to computing resources, which are discussed in next section.

The main phases and actions involved in each phase of grid scheduling has been discussed in [41]. The three main phases involved in grid scheduling are:
a) Resource discovery phase
b) System selection phase
c) Job execution phase

**a) Resource Discovery Phase**

In the resource discovery phase, the grid resources with their attributes available for executing grid applications are discovered using grid information services. The activities involved in resource discovery phase are - Authorization and authorization, job requirement definition and filtering to meet the minimal job requirements.

1) Authentication and resource authorization:
This activity involves authentication of the users submitting and job and checking the authorization of the users to access the grid resources.

2) Grid Application (Job) requirement definition:
Different grid jobs require different type and amount of resources and their QoS requirements vary. To perform the efficient scheduling of grid jobs, users should specify the resource requirements for execution of grid jobs. Users specify the grid job resource requirements in the form of job description file. The job requirement definition includes all the static details like operating system, processor speed, architecture and dynamic details like secondary storage required, minimum RAM required or network bandwidth needed.

In traditional cluster systems like PBS or LSF, the users specify the basic job submission details in their job submission scripts. In condor matchmaking, it is specified in submitted ClassAd of the job.

3) Requirement Filtering
It involves identifying the eligible grid resources as per the job requirement definition and eliminating the resources that do not match the static and dynamic job requirements. This filtering is useful to grid scheduler for taking scheduling decisions.
Chapter 2: Literature Survey

b) System Selection Phase

Given the grid application and set of eligible resources, the system selection decision is required to be taken to schedule the grid job. This phase involves:

4) Dynamic Information Gathering:

In grid system, it is important to have dynamic information about the status of various grid resources like CPU load, available main memory etc. so that effective scheduling decision can be made. Grid information services (GIS) and local resource managers can provide such information based on the grid configuration.

5) Scheduling and System Selection

This is the most important decision taken by the grid scheduler based on the above steps of activities performed. Various scheduling approaches are used for system selection like condor matchmaking, multi-criteria scheduling or using heuristic algorithms.

c) Job Execution Phase

6) Resource reservation in advance

Resource reservation is optional activity, but for certain grid applications with QoS constraints like job completion deadline, it is important to have support for advance reservation of grid resources.

7) Job Submission

After the resource selection, the grid scheduler submits the application to the selected grid resources. Job submission to remote nodes may involve file staging i.e. uploading the input data to the remote nodes. Various standard approaches and protocols like
Chapter 2: Literature Survey

GRAM (Grid Resource Allocation and Management) by Globus Consortium, DRMAA (Distributed Resource Management Application API) by Open Grid Forum have been developed for submission and control of grid jobs on remote resources.

8) Job Monitoring and Progress

Most grid applications are long running batch jobs. The grid scheduler or users must monitor the execution and status of grid jobs in a periodic manner. The jobs which are in waiting state for long periods can be rescheduled to other resources. The grid user/scheduler needs to be updated about the status of remotely running jobs, and if the job has failed, it needs to be submitted again.

9) Job Completion and Clean-up tasks

After the successful execution of remotely submitted grid job, clean up and completion activity needs to be undertaken by the grid scheduler. Job completion and clean-up activities can be downloading of result files, aggregation of results from processing nodes by grid scheduler, releasing the remote resources, deletion of temporary files on remote resources and updating the status of grid application and notifying the grid users.
Chapter 2: Literature Survey

General grid scheduling architecture

General grid scheduling architecture involves various layers of grid schedulers. They can be classified as local schedulers and global or grid level schedulers. General architecture of grid scheduling framework is as shown in below figure.

Grid resource management and scheduling involves

- Application requirements into resource requirement
- Resource discovery based on attributes
Chapter 2: Literature Survey

- Job submission
- Resource matchmaking with application requirements
- Resource negotiation and allocation based on QoS requirement
- Service Level Agreements (SLA) between grid system and its clients
- Scheduling and co-allocation
- Resource monitoring
- Resource reservation capability and
- Performance prediction capability

Traditional applications execute in static and confined environment with fixed assets. But the Grid environment is somewhat dynamic consisting of heterogeneous resources. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated [41].

Various merits and de-merits of different types of grid scheduling algorithms is done in [93].
2.6 Study of Grid Scheduling Algorithms and Existing Models

Different grid scheduling approaches and algorithms have been discussed in [94] and [95].

A Hierarchical taxonomy for scheduling algorithms has been proposed in [92]: local v/s. global, static vs. dynamic, approximate vs. heuristic, distributed vs. centralized.

A Comparison among grid scheduling Algorithms for independent coarse-grained tasks is done in [96].

A distributed computing scheduling taxonomy is presented in [97].

Scheduling and State estimation aspects of distributed scheduling are presented in [98].

Static versus Dynamic

In static scheduling, resources are assumed to be static and static resource information is used in scheduling decision. At the time of scheduling the application, all the information about the resources and tasks in the grid system is assumed to be available [99]. Static scheduling is not adaptable as it does not consider abnormal situations like resource failure or network failure while taking scheduling decisions. In dynamic scheduling, dynamic grid resource information is gathered and it is used to perform the allocation of grid tasks to grid resources. It is useful when jobs arrive in real time mode and it is not possible to determine the execution time in advance.

Approximate versus Heuristic

Approximate algorithm can be useful for solving NP-hard problems where it is time consuming to search the entire solution space. Such algorithms use evaluation metric using formal computational models to find a satisfactory solution. Such techniques are useful to find a near optimal satisfactory solution within time constraints satisfying
evaluation criteria. A new objective function based approximate algorithm is proposed in [100].

Heuristic Scheduling

Heuristic scheduling algorithms are useful when the traditional scheduling techniques fail to find optimal solution. “Given an application consisting of n jobs, either independent or dependent, and a set of heterogeneous resources, the problem of mapping the n jobs into the m resources subject to some criteria is a well-known NP-complete problem [101]. Grid scheduling being a NP-hard problem, various heuristic based scheduling techniques can provide optimal solution within time and space constraints. For such problems, no known algorithms are able to generate the optimal solution within polynomial time, i.e. when the input data grows in size, the execution time of such algorithm increases exponentially. That is, the time required to solve the problem using any currently known algorithm increases very quickly as the size of the problem grows.

In grid scheduling with QoS - multiple constraints and optimization criteria are required to be fulfilled, which makes it a complex and NP-hard problem. To solve such problems, heuristic approaches needs to be used. Heuristic scheduling algorithms provide more adaptive and efficient result considering the heterogeneity and dynamicity of grid resources, applications and different types of failures in grid system. Various heuristics based scheduling techniques have been proposed in [102].

Various local and population based heuristic search algorithms have been proposed in the literature. Methods in this approach iterate over different solutions by exploring the solution space to find the best feasible solution fulfilling the constraints [103]. Different heuristic grid scheduling algorithms based on this approach are Genetic Algorithm, Hill Climbing, simulated annealing, tabu search, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) [104].
Distributed, Centralized and Hierarchical

In computing grid architecture, there can be local schedulers and meta-schedulers coordinating with multiple local schedulers managing local grid resources. In such a scenario, scheduling decisions are taken at global level as well as local level in coordinated manner. In centralized scheduler, the scheduling decision is taken by centralized meta-scheduler which coordinates with local distributed schedulers. In distributed scheduling, the scheduling decision is taken jointly by multiple distributed schedulers in grid system. Distributed scheduling has the advantage of being more fault-tolerant, scalable and efficient as compared to centralized scheduler in grid scenario. The distributed scheduler can be cooperative or non-cooperative. Cooperative schedulers work in coordination with each other to fulfill the overall grid system wide objectives. Non-cooperative schedulers take scheduling decision as independent entities and optimize their individual objects e.g. application level schedulers try to optimize the execution of individual application without having concern for overall grid system.

Adaptive Scheduling

As grid is a dynamic system where the state of the system keeps changing, the scheduling algorithm has to be adaptive to the changing state. Because of resource autonomy, performance and status of the grid resources are dynamic. In adaptive scheduling, algorithms and parameters used to make scheduling decisions change dynamically according to the previous, current and future resource status [105].
SETI@home which is internet based voluntary desktop scavenging grid computing project, gives priority to local jobs. The Grid task executes only when no local jobs are running [26].

The Condor [106] system uses a matchmaker that maintains a pool of ClassAds representing candidate resources and then matches each incoming request ClassAd with all candidates, returning a highest-ranked matching candidate [107]. After matchmaking, the condor broker does not interfere and the user works with resources directly [108]. But in our work, the broker manages the activities till the results return to the user, to maintain QoS and adaptability.

**Backfilling**

Backfilling is an effective strategy for scheduling parallel tasks efficiently. In systems of parallel tasks, the job at the head of the queue is often delayed because of a lack of available resources. However, there are often less demanding jobs in the queue which could be executed earlier without delaying the job at the head. EASY backfilling follows this strategy [109]. If the resources for the job at the head of the queue are expected to become available at time t, then the easy backfilling strategy allows a job with expected length l, where l < t, to be executed immediately, rather than waiting, assuming its required resources are available. This effectively fills the idle bubbles in the schedule and results in a shorter overall completion time.

**Makespan Minimizing Heuristics**

The makespan of a set of tasks is the time taken between the start of the first grid task and the completion of the last grid task. Finding the minimal makespan is nondeterministic polynomial-time hard (NP-hard) [110]. As such, research in operations and heterogeneous computing has resulted in several heuristic approaches to minimizing the makespan [111]: Min-min, Max-min, Sufferage and XSufferage.
**Min-Min**

Heuristic algorithm (The Min-Min algorithm) for scheduling m jobs on n machines is proposed in [111]. The min-min heuristic assigns the shortest task to the resource that will execute it the fastest. The algorithm iteratively assigns jobs to machines by considering jobs not yet scheduled and computing their expected Minimum Completion Time (MCTs). The Min-Min algorithm does not take into account the QoS issue in the scheduling. In some situation, it is possible that normal tasks occupies machine that has high QoS resources. This may increase the delay of QoS tasks or result idle of normal machines and uneven load balancing in the system.

**Max-Min**

The max-min heuristic is similar to min-min, but instead prefers to run the longest tasks early. In the situation where there are many short tasks and few long tasks, then the min-min heuristic results in low efficiency and suboptimal overall completion time. Because min-min postpones the long tasks, they are left running on a small proportion of the processors, leaving the remaining processors to idle. The overall completion time is therefore dramatically increased for such sets of tasks.

**Condor Matchmaking**

Matchmaking [106] is the resource allocation framework employed by Condor and Condor-G [112]. Condor is designed for high throughput computing (cluster environment) in a controlled local network environment. In this strategy, each consumer (job) and provider (resource) provides a classified advertisement (ClassAd), which contains the entity's description and characteristics. However, matchmaking itself does not result in any resource allocation optimizations; rather, its value is that it provides a framework within which such optimization heuristics can be implemented.
Chapter 2: Literature Survey

**AppLeS**

The scheduling algorithm in AppLeS [113] focuses on efficient co-location of data and experiments as well as adaptive scheduling, but does not consider QoS in scheduling.

Widely used scheduling strategy in grid computing is based on policy and set of rules for resource use and assignment of task or job to a specific node depending on load balance and need of resource [112]. Various task and application scheduling approaches for grid have been published in the literature [92]. Various performance modelling strategies for predicting execution times of parallel applications has been proposed in [114]. Economic based scheduling heuristic for utility grids has been proposed in [115], which has been validated using GridSim simulator. The above approaches do not consider the inherent nature of parallel application in efficient scheduling of high performance scientific parallel applications. Buyya et. al. [116] has proposed cost-time optimization algorithm and Abramason et. al. [117] has proposed Nimrod/G. These systems are useful for scheduling parameter sweep parallel applications with deadlines and cost constraints. They do not consider individual user’s different QoS requirements.

Various Economic Models for Resource Management and Scheduling in Grid Computing has been proposed in [119] [120].

**Sun N1 Grid Engine**

Sun Grid Engine is an open source distributed resources management (DRM) system that maximizes resource utilization and workload throughput while adhering to resource allocation policies and business rules [120]. Sun N1 Grid Engine 6.1 accepts grid jobs submitted by users. “The software uses resource management policies to schedule jobs to be run on appropriate systems in the grid. Users can submit multiple jobs at a time without being concerned about where the jobs run. The main tasks that Sun Grid Engine performs can be categorized as follows [121]:
Chapter 2: Literature Survey

- Distributed job scheduling
- Resource balance and run-time management
- Authentication and authorization
- Resource and jobs monitoring

Four types of hosts are defined for the working of Grid system [67]:

1. **Master host** (to manage the nodes, addition and deletion of execution hosts)
   The master host is basically responsible to maintain all the information about the cluster and take decisions on jobs scheduling, acting as the director of the grid.

2. **Execution host** (provide processing power for executing the jobs submitted)
   The execution hosts are nodes that have the permissions to run SGE jobs using a set of SGE queues, they represent the computational core of the cluster, sharing the concrete resource and providing CPU power, storage and memory capabilities.

3. **Administration host** (admin console)
   Administration host provides the administrative functionality like monitoring and controlling the remote jobs.

4. **Submit host** (allow job submission to the grid)
   The submit hosts are those nodes allow to schedule batch jobs to the cluster, these kind of jobs are all the tasks that don’t need interaction from the user, therefore the scheduling consists only on submit the job and save the output to a file rather than manage an interaction between the user and the job in the case of for example programs with a graphical user interface.

A compute grid with heterogeneous resources was implemented using Sun N1 Grid Engine using Linux and windows machines participating in the grid. In the implemented grid architecture, Linux nodes serve as execution hosts providing processing power for executing the grid jobs, master host (controlling and administration) and windows nodes are used for job submission and result collection [122]. Jobs are submitted from window hosts and the computational processing is done on Linux hosts and the results are returned to the windows host.
Batch jobs can be submitted to N1 grid engine using qsub command and QMON GUI. QMON provides a job submission dialog box and a Job Control dialog box for the tasks of submitting and monitoring jobs. Grid jobs can be scheduling using different high-level scheduling policies – Functional, Urgency, Share-based, Override. Various experiments by were performed by executing different types of computational jobs in the grid engine and based on experience various limitations of N1 Grid Engine are as below:

• If the master host is down, it affects the working of whole grid system. Dependency on the central master host machine affects the concurrency and scalability of the parallel jobs submitted.
• Not possible to integrate the Sun grid with other grids and clusters created using Globus or Condor.
• Complex configuration and software installation required on each machine which has to participate in the grid (even as job submission host or execution host).
• Windows machines are difficult to integrate in the grid as it requires complex configuration and installation of additional software (Windows services for UNIX, SFU). Windows machines can serve as only job submission host and cannot server as processing nodes. The installation of Services for UNIX (SFU) requires a good administrative understanding of the Windows platform and its integration into a UNIX environment. Additional requirements of same username mapping on all machines and remote NFS mounts.
GridWay Metascheduler of Globus Toolkit

GridWay, on top of Globus Toolkit services, enables large-scale, reliable and efficient sharing of computing resources (clusters, computing farms, servers etc.), managed by different DRM (Distributed Resource Management) systems, such as PBS, SGE, LSF, Condor..., within a single organization (such as an enterprise grid) or scattered across several administrative domains (partner or supply-chain grid) [123].

It provides meta-scheduling that provides advanced scheduling functionality to grid users, application developers, and managers of globus infrastructure. As shown in above figure, it provides support for various local resource management systems like PBS, Condor, SGE etc. within single organization or across multiple domains. It provides unified single point of access to all the grid and cloud resources managed by various grid middleware and cloud providers. It has below features [124]:

- Flexible and extensible architecture
- Improved efficiency and robustness
- Advanced scheduling capabilities
- Information drivers to interface MDS2 and MDS4
- Execution drivers to interface pre-WS GRAM and WS GRAM, even both simultaneously
- Transfer drivers to interface GridFTP and RFT
- Full support for C and JAVA DRMAA GGF standard
- DRM-like commands to submit, monitor, synchronize and control jobs; monitor Globus
- Resources and users; and extract Grid accounting information
Figure 2.8: Architecture of GridWay Metascheduler [123]
2.7 Classification of Grid Applications

Various resource intensive applications in different fields like medical healthcare, bioinformatics, geological and climate predictions, mathematics, life sciences, business decision support, physics among others are candidates for execution on grid platforms. Different distributed applications in different domains have different characteristics and application flow. Different categories of grid applications have been discussed in [125]:

Serial flow

The grid application consists of multiple jobs and the jobs are required to execute in serial order. Each successor jobs required the data produced by the predecessor job in the application. Each job is independent and can be executed on available matching resources. The ability to use the best available grid resource and not having dependency on particular resource increases the overall applications performance, availability and reliability. E.g. if a grid application is having the jobs A, B, C and D, then the serial execution will required the jobs to executed in serial order of A, B, C and D.

Parallel flow

![Grid parallel flow application](image)

Figure 2.9: Grid parallel flow application
Chapter 2: Literature Survey

Compute intensive applications with several jobs that can be executed in parallel falls under this category. Such applications are high performance computing applications that use MPI (message passing interface) library for communication among parallel jobs. Such applications have fine-grained parallelism, where each task depend on the output of other tasks and tasks are required to communicate by message passing. Criterion for parallel application is that it can be broken into small tasks, and each task can be assigned to different processor to speed the overall application. In such applications, from an initial or master job A, several worker jobs are launched on remote grid nodes like B, C, D and E as shown in Fig. 2.1. Each worker jobs executes independently to perform the work assigned by master, and sends the intermediate result back to master job, which then aggregates the intermediate results from all worker jobs and calculates the final result.

**Parameter Sweep Application**

Parameter sweep applications are common in various science, engineering, finance and mathematical simulations. Such applications are embarrassing parallel applications where each task can be executed independent of other tasks with no inter communication, E.g. Monte Carlo simulations or processing of large number of medical images for detection of anomaly. In such applications, same job is required to be executed large number of times, each time with different data sets. Such applications are high throughput applications with combination of task and data parallel model - single program multiple data (SPMD) model.

**Network workflow application**

In such applications, different jobs have complex interdependencies between them and have data flows between them in both directions. Also, the different jobs need to be executed in particular order as per the workflow and logic of the application.
In the example of network flow application shown in above figure, jobs are having interdependencies between them.
2.8 Quality of Service in Grid Computing

Quality is a non-functional attribute and means different things to different people, domains, systems and applications. The term - Quality of Service (QoS) has been studied and researched in various domains by various authors and researchers [126][127][128][129]. The general understanding of QoS means providing priority based desired levels of service performance as per QoS metrics rather than “best effort” approach. QoS also encompasses conformance to desired specification with measurable characteristics and ability to meet end user expectations and needs.

Providing QoS oriented grid resource management and scheduling is a constraint satisfaction and time consuming optimization problem.

Related work done concerning Quality of Service in Grid Computing is summarized as follows:

Grid resource management system performs the request handling, resource allocation, and scheduling and job execution to maximize some metric of aggregate Quality of Service (QoS) [130].

In the paper [131], the authors focus an application-level scheduling system for parameter sweep applications, which are applications composed of a set of independent tasks and presents a task graph model for enterprise grid applications.

The importance of research on grid Quality of Service (QoS) and QoS problems peculiar to the grid environment has been discussed in [132].

In [133], architecture of the Virtual Resource Manager and its main QoS management features like run-time responsibility, co-allocation, and fault tolerance have been proposed.
Chapter 2: Literature Survey

Various grid QoS aspects and metrics are discussed in [134]. QoS mechanisms which can be applied to various layers of Grid architecture are discussed.

QoS guarantees in grid computing based on advance reservations were investigated in [135].

2.9 Agent Based Grid Computing

Software agent is a computer program representing logical entity and having the capability of autonomous behavior, ability to communicate with other agents and take decisions. Software agents are recognized as a powerful high-level abstraction for the modeling of complex software systems [136].

Software agents which can migrate from one node to another with autonomous behavior are called as mobile agents. A mobile agent is a software agent which is composition of computer software and data which is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer [137]. When the term mobile agent is used, it refers to a process that can transport its state from one environment to another, with its data intact, and still being able to perform appropriately in the new environment.

Agents are [138]:

(i) clearly identifiable problem solving entities with well-defined boundaries and interfaces;
(ii) situated (embedded) in a particular environment—they receive inputs related to the state of their environment through sensors and they act on the environment through effectors;
(iii) designed to fulfill a specific role—they have particular objectives to achieve and have particular problem solving capabilities (services) that they can bring to bear to this end;
(iv) autonomous—they have control both over their internal state and over their own behavior; and
(v) Capable of exhibiting flexible problem solving behavior in pursuit of their design objectives—they need to be both reactive (able to respond in a timely fashion to changes that occur in their environment) and proactive (able to opportunistically adopt goals and take the initiative).

Distributed system like Grid can be represented as a set of software agents interacting and working together for providing grid functionalities. Grid agents are able to behave autonomously, intelligently, self learning capability from the environment and adaptability. The agent grid computing middleware systems provides the coordinating middleware distributed agents which coordinates a user job over remote computing nodes in a decentralized manner.

The main focus of agent in a grid environment is to maintain high availability and dynamic balancing of distributed computing resources to a parallel computing job. For this purpose, agents are assigned to each process in the particular job, monitors its execution under different configurations, reviews the statistics periodically, moves it to a
machine with lighter load and resumes the execution from the last snapshot upon accidental crash.

Agents have the capability to adapt themselves in the available grid environment. Agents provide robust and scalable services to achieve the goal of grid. Resource management is an important infrastructural component of a computational grid.

Considering the above discussion, above figure above shows that adopting an agent-oriented approach to grid system engineering means decomposing the problem into multiple, interacting, autonomous components that have particular objectives to achieve and are capable of performing particular services. The key abstraction models that define the agent-oriented mindset are agents, interactions and organizations.

Multi-agent systems can be developed using JADE (Java Agent Development Environment) [139].

The following issues make grid resource management system (RMS) complex and crucial: site autonomy, resource heterogeneity, and difference in resource usage, scheduling policies, security mechanisms, and requirement of interfacing and interoperation with local resource management systems, requirement for the negotiation between resource users and resource providers [140].

Various characteristics and advantages of agents are discussed in [141]. The above work indicates the fault tolerance and adaptable nature of agents and their suitability for solving distributed computing problems.

The convergence of agent and grid systems, agent systems requiring robust infrastructure and Grid systems requiring autonomous, flexible behaviors is proposed in [136]. Various Multi agent systems and technologies are discussed.

A computational grid can be viewed as a multi-agent system, in which computational devices on the grid are intelligent agents with a significant degree of autonomy and are
capable of performing their own tasks, sharing their resources across the grid and communicating with other agents on the grid.

In [143], agent based abstractions for modeling grid applications and architecture for grid computing is considered. The architecture enables routing and handling of agent communication language (ACL) messages.

Intelligent agents can play an important role in helping achieve the grid vision [144]. In this paper, model for agent-based grid computing has been proposed from the implementation point of view.

Solutions have been proposed in [145] for deployment and application of Grids in an open environment. This work shows that integrating peer-to-peer computing and multi-agent technologies leads to improved scalability, efficiency, flexibility, and robustness in open environments like grid, compared with conventional Grid computing architectures.

Multi-agent based peer-to-peer grid computing architecture is proposed in [146]. The above architecture does not consider QoS issues in resource management and scheduling.
2.10 Outcome of Literature Survey and Gap Analysis Findings

Grid Resource Management and Scheduling is the central and complex part of any grid implementation. The following issues make RMS complex and crucial: site autonomy, resource heterogeneity, and difference in resource usage, scheduling policies, security mechanisms, and requirement of interfacing and inter operation with local with local resource management systems. In the Matchmaking phase, the choice of the best pairs of jobs and resources is a NP-hard problem, which requires heuristic scheduling.

Various grid resource management and scheduling models have been published in the literature. But, they lack the holistic resource allocation and scheduling mechanism in grid computing to provide differentiated QoS levels to different jobs. There is a need for QoS oriented grid model which provides reliable, scalable, fault tolerance and adaptive grid resource brokering and scheduling for efficient execution of compute intensive applications.

Various issues in grid scheduling are - Heterogeneity and autonomy of the resources, Performance dynamism of environment where resources are not dedicated and constantly changing, Heterogeneity of application requirements (CPU, I/O, memory, and/or network intensive), Differing goals (resource providers and consumers have different objectives and strategies).

Grid middleware like Globus [48] [49] and Sun N1 grid engine [50], Condor do not provide any deterministic quality of service to grid clients, and they rely on a centralized scheduler which lead to problems of scalability, adaptability and reliability of the grid system. To provide support for real-life applications, computational grid must provide users with non-trivial quality-of-Service (QoS) [51] covering metrics such as deadlines, reliability, security and budget associated with service invocation.

Grid Resources managed by different grid middleware like Sun grid engine middleware and Globus toolkit are not interoperable. Most Grid middleware and environments
provide best-effort scheduling, do not provide differentiated services or QoS based resource provisioning. There is a need for holistic resource allocation and scheduling mechanism in grid computing to provide differentiated QoS levels to different jobs providing scalability, reliability and fault-tolerance. The ‘best-effort’ approach is inappropriate if grids are to be used as the infrastructure for “real world” commercial applications and complex scientific applications like simulations and modeling.

Quality of Service (QoS) in Grid refers to providing differentiated services to different grid applications by prioritizing them and maintaining desired level of service performance. It also includes user centric priority based resource management and scheduling to fulfill user specific QoS parameters [52]. User specified QoS parameters are: job resource requirements with associated quality levels, resource reservation, execution deadline and budget. The grid QoS problem is to find the QoS based optimization of grid resource management and scheduling based on above specified QoS parameters by grid applications.

The QoS oriented grid should abstracts the end users from the underlying complexity of low level grid middleware and provides a user friendly interface to end-users to submit jobs with associated QoS parameters. Grid resource management and scheduling should consist of following tasks to provide QoS support [53]:

- The grid resource broker extracts the resource and QoS requirements from the user
- Resource discovery, which generates a list of resources with their attributes
- Matchmaking (matching resources to applications) to find the optimum set of resources minimizing the execution time and fulfilling the QoS requirements
- Resource reservation (advanced or immediate) based on the job requirement
- Allocating and co-allocating resources
- Job Scheduling and submission of jobs (tasks) to grid resources