Chapter 2

Literature Survey

Grid computing has emerged as an incredible technology to provide huge amount of power to large-scale scientific, innovative applications, and in some cases, high-performance orientation and e-commerce projects.

The deployment of Grid systems involves the efficient management of heterogeneous, geographically distributed and dynamically available resources. As Grid comprises of resources, Grid resource management is the fundamental task in Grid computing. The Grid resource management system is required to perform resource management decisions which include resource provisioning and scheduling, while maximizing the QoS metrics delivered to the clients when jobs have QoS constraints.

This chapter discusses the Grid resource management systems along with the description of Grid resource provisioning and scheduling requirements. It also discusses the key background information to facilitate a better understanding of resource provisioning and resource scheduling in Grid environment. A comparative study and analysis of the existing Grid schedulers and scheduling in Grid middleware has been done. Further, Grid scheduling heuristic approaches have been discussed and compared along with the study of QoS in Grids. The last section highlights the limitation of the existing approaches and finally lists down the objectives of the thesis.

2.1 Grid Resource Management Systems

Grid computing infrastructure promises to provide the facility of resource sharing in distributed systems in a flexible, secure and coordinated fashion [2] [62]. The main vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically as and when required, similar to a power Grid which supplies electricity to the end users [63]. Grid computing enables sharing of loosely coupled resources and services required by different applications on a large-scale. Resource is the basis of the Grid and managing the resources in Grids is a tedious task. Thus, resource management is the core of a Grid system as defined by Jyotishman Pathak et al. [64]. Resource management describes all aspects of the processes: locating a capability, arranging its use and utilizing its monitoring state. It also includes the efficient use of computing and storage resources. The basic function of a resource management system is to accept requests for resources and assign specific machine resources to a request from the overall pool of Grid resources for which the user has the access permission. Resource management includes various services such as launching a job on a particular resource, checking its status and retrieving results when the job is complete as descried by Li M. and Baker M in their book [55]. This mechanism can be further described as follows:

i. Resource Information Dissemination: It involves collection of information about all the resources required for the execution of application in the VO.

ii. Resource Discovery: Resource discovery is the process of matching a query for resources, described in terms of required characteristics, to a set of resources which meets the expressed requirements [59]. Authorization filtering and job requirement knowledge to meet the minimal job requirements' needs to be accomplished. Various approaches like agent based resource discovery, query based resource discovery, peer to peer approach, parameter based approach and QoS based approach can be used for resource discovery in Grid environment.

iii. Resource Provisioning: Resource provisioning allows users and providers to access the specified resources as per their availability in the virtual environment created in Grid infrastructure. Various strategies which can be used for resource provisioning in Grid environment includes static and dynamic resource provisioning, agreement based resource provisioning, model based resource provisioning and application based resource provisioning, etc.

iv. Resource Scheduling/Job execution: Resource scheduling allocates jobs on the provisioned resources and performs the task of job execution [65]. A variety of techniques and algorithms including static and dynamic scheduling, heuristic and approximate scheduling, cooperative and non-co-operative scheduling are utilized for job scheduling and execution. Other scheduling techniques like workflow scheduling, task level scheduling and adaptive scheduling are also explored for the Grid environments [26].

v. Resource Monitoring and Re-Scheduling: Resources must be monitored to track the status of allocated resources, available resources and required resources for application execution. In case a job fails to perform or a resource bottleneck occurs, then the resources are rescheduled. Static, dynamic and workflow monitoring are being done in the Grid environment.

With Grid becoming a viable high-performance alternative to the traditional supercomputing environment, various aspects of effective Grid resource utilization are gaining significance. With its multitude of heterogeneous resources, resource provisioning and proper scheduling in the Grid is required for improving the performance of the system [66]. As the Grid scales up, resource management complexity also increases and efficient resource management techniques are desired. Ramrez-Alcaraz J.M. et al. defined that resource management is one of the key components of Grid computing where QoS based resource provisioning and resource scheduling are challenging tasks [67]. To increase the efficiency of resource management systems, there is a need to concentrate on resource provisioning and scheduling. Resource provisioning and scheduling are the key components for Grid resource management besides numerous other components. In resource management systems, one of the key challenges is to design resource provisioning policies and resource provisioning based scheduling algorithms which would allow users to carry out their jobs by transparently accessing autonomous, distributed and heterogenous resources. Foster, I. et al [27][2] defined that resource provisioning allows the users and providers to access the specified resources according to the their availability of the resources in VOs. Grid scheduling makes scheduling decisions involving allocating jobs to resources over multiple administrative domains [65]. The main aim of the current resource provisioning and resource scheduling is to provide Grid's users and non- users with privileges of using resources in advance level while satisfying the QoS parameters. Deelman, E. suggests that resource scheduling can be done subsequently after resource provisioning as the user desires to discover the resources according to the requirements of the job.

Then resource mapping is done to that particular job. Resource provisioning is particularly useful for Grid scheduling because most often Grid scheduling is done on the best-effort basis in a Grid environment and due to this, overheads of Grid scheduling like inter-independent tasks can be very costly and time consuming [68]. For example:- If there are two jobs in a queue and the first job starts execution then the second job will not be released to the local resource manager until the first job finishes. But in case of resource provisioning, resources are provisioned and then the second job is released to the local resource manager immediately after the completion of the first job. Thus, queuing time delay will be less through resource provisioning.

The next section presents the requirements for the resource provisioning and scheduling in Grids.

2.1.1 Grid Resource Provisioning and Scheduling Requirements

There are various provisioning and scheduling requirements for Grids. Few of them have been identified by Dusseau A. C. A. in [69] as:
Efficiency: Resource provisioning provides the facility to minimize the Grid overheads. It requires efficient management and scheduling of the resources based on fair policies.
Efficient Resource Usage: It reduces the wastage of the resources. Jobs that are waiting for events (e.g., disk or user I/O, network latency, CPU usage, processor) should hand over the processor so that they do not waste resources.
Fair Allocation: The amount of resources allocated to each user should be independent of the number of jobs each user runs and provisioning of resources should also be fair.
Adaptability and scalability: A smart scheduler adapts as per the resources, i.e. even when the resources join or leave (dynamically), it manages the resources and jobs' execution process efficiently.

After discussing the Grid resource management systems, its importance and the requirements of resource provisioning and scheduling in the Grid, next section would focus on the resource provisioning and its existing practices in Grid computing.

2.2 Resource Provisioning

The resource availability in Grids is generally unpredictable due to the autonomous and shared nature of the Grid resources and stochastic nature of the workload resulting in a best effort Quality of Service [70]. Singh G. et al. defined that due to large-scale, shared and autonomous nature of the Grid resources, modeling the resources availability in discrete units that can be provisioned by the user is a more appropriate approach [71]. For resource provisioning, QoS parameters need to be defined and provided to the user. Guaranteeing QoS in a non- deterministically shared heterogeneous environment like Grid is a challenging task. Provisioning is slightly more complex than queuing in the way that it requires users to make more sophisticated resource allocation decisions [72]. As per Gideon Juve and Ewa Deelman, resource provisioning can be classified as follows:

Static provisioning: In static provisioning, application allocates the resources for job completion which are required for job processing. After the job completion, application releases the resources [72]. This method assumes that the number of resources required is known or can be predicted in advance.

Dynamic provisioning: In dynamic provisioning, system allocates the resources at run time. It allows the pool of available resources to grow and shrink according to the changing needs of the application [72]. Dynamic provisioning does not require advance knowledge of the resource but it requires the information about policies for acquiring and releasing the resources.
Resource provisioning can be implemented in the following ways:

Agreement based resource provisioning: The agreement-based management model allows the VO-level resource brokers to provision resources ahead of the execution of the applications by entering into agreements with the resource providers about the guaranteed availability of desired resources for a mutually agreed upon time-frame and cost [70].

Model based resource provisioning: Model-based approach to provisioning is multiple resource based that interacts in complex ways. In case of model-based resource provisioning, internal models capturing service workload and behavior can enable the utility to predict the effects of changes to the workload intensity or resource allotment [73].

Application based Resource Provisioning: Application based resource provisioning allows the user to control the scheduling and execution of the application on the provisioned resources [71]. This in turn enables the estimation of the application performance prior to its execution. It also provides the facility to explore the space of resources to be provisioned and optimized for performance without worrying about external factors such as the workload of the resource or the policies of the resource provider to the user or workflow manager. Application based resource provisioning enables the adaptation of the application on the provisioned resources in order to achieve a certain level of performance. It also allows the execution of applications that require co-allocation of resources or have other constraints on the resource requirements.

AAA Policy based Provisioning: The main function of AAA policy based provisioning is to incorporate AAA (Authentication, Authorization and Accounting) policy into path computation resource allocation and signaling functions. It requires high-level association of policy with users as well as lower-level association of policy with actual network elements at a fidelity sufficient to implement meaningful policy based resource allocations [74].

Resource provisioning and scheduling are the core components of resource management in Grid computing. There has been little emphasis on resource provisioning for Grid resource management as described in the next section. The categorization of resource provisioning with QoS and without QoS has been discussed in the following section.

2.2.1 Resource Provisioning- without QoS

DRAGON [75] and GLARE [76] frameworks provide provisioning in heterogeneous Grid environments. As per DRAGON framework, network infrastructure is deployed, that allows dynamic provisioning of network resources in order to establish deterministic paths in direct response to end-user requests. It also allows advanced e-science applications to dynamically acquire dedicated and deterministic network resources to link computational clusters, storage arrays, visualization facilities, remote sensors and other instruments into globally distributed and application-specific topologies. GLARE provides distributed registries for activity types, activity deployments and services that perform registration, provisioning, monitoring and automatic deployment of new activities on different Grid computers in VO. Murphy et al. [77] have designed the Virtual Organization Clusters (VOC) architecture that provides a mechanism by which each VO may have its own dedicated clusters on Grid sites. It can be dynamically expanded, contracted, instantiated or terminated in response to the size and number of jobs submitted by the user. Dynamic provisioning has been done in VO.

Yu et al. introduced the Adaptive Online Job Provisioning (AOJP) that aims to provide co-scheduling of computing and network resources in an optical Grid infrastructure. With the help of mixed integer liner programming, AOJP has solved the data aggregation problem which focuses on how to schedule network resources to fetch data from multiple sources to a single sink [78]. Byun et al. proposed an architecture to enable on demand resource provisioning and have developed a universal factory service that provides a dynamic Grid service deployment mechanism and a resource broker called door service for service providers [79].

Nau et al. have introduced an approach to allow heterogeneous applications to run together in a shared hosting platform, dynamically sharing the platform's resources [80]. Vazquez et al. addressed dynamic provisioning problem as interoperability, dynamic growth and enforcement of a budget by designing the architecture for an elastic Grid infrastructure [81]. Feasibility solution by harnessing resources of the Tera Grid, EGEE and Open Science Grid infrastructures through a single point of entry for the computational resources has been given for Grid infrastructures. AOJP and these solutions of dynamic provisioning do not support the QoS parameters for online job provisioning. QoS management in Grid computing has become an active area of research now-a-days. However, there are several applications that need to obtain the results for their tasks within time, hence they cannot wait for resources to become available [60]. Therefore, it is necessary to provide QoS at the time of resource provisioning.

2.2.2 Resource Provisioning- with QoS

OGSA based service ecosystem [82], a resilient infrastructure for services' provisioning on demand has been developed. OGSA based service ecosystem framework considers the status of Grid and customer requirements automatically to achieve the QoS parameters e.g. performance and reliability. This work provides services' provisioning using migration concept for commercial Grids and not for non-commercial Grids. Keller et al. presented strategies for Grid service provisioning to handle resource failures during Service Level Agreement (SLA) negotiations. Risk aware failure management has been considered in this work in the form of QoS but other important QoS parameters as time and cost have not been discussed [83].

Filali et al. [84][85] used the adaptive resource provisioning schema to achieve resource utilization in the best manner satisfying the required QoS by maximizing the network utilization, minimizing request blocking probability and maximizing the provider's revenues. Local optimization heuristic and global optimization heuristic have been proposed by using the optimization techniques. An optimization model that enables the simultaneous allocation of interrelated resources as CPU and bandwidth for Grid applications has been proposed.

Kee et al. [86] proposed a high performance distributed computing resource management paradigm that defines a network of logical machines across time and space. A resource management system to provide resource provisioning, resource abstraction via resource programming over large-scale distributed resources has been used in this work. Virtualization for timely targeted applications in the Grid environment has been considered.

Foster et al. [87] described a General-purpose Architecture for Reservation and Allocation (GARA) that supports flow-specific QoS specification, immediate and advance reservation and online monitoring and control of both individual resources and heterogeneous resource ensembles.

QoS-GRAF framework [88] for QoS based Grid allocation with failure provisioning has been designed by Dasgupta et al. By considering SLA based service differentiation and failure provisioning, a linear relaxation based algorithm has been designed to improve the revenue.

Iosup et al. [89] have discussed about on demand resource provisioning for dynamic environment with worldwide sharing resources. A scheduling policy has been designed to solve the problem of resource shortage with heavy workload but security at the time of provisioning has not been considered for data sharing. Brocco et al. have proposed a flexible Grid service provisioning framework composed of two functional layers: a self-organized peer-to-peer overlay management layer, and a Grid services interface layer [90]. It has been designed to provide an adaptive, reliable and scalable communication platform for service provisioning in Grid network. Singh et al. [71][70] have presented a provisioning model for resource provisioning where Grid sites provide the information about resource availability in the form of time slots. The main aim of the model is to identify the subset aggregate resource availability so that cost and make span are minimized. A multi-objective genetic algorithm to find resource plan that corresponds to the Pareto optimal set has been used in this work. Raicu et al. [91] have proposed a dynamic resource provisioning architecture using the existing system *falkon*. Here allocation and de-allocation policies are presented and these policies are evaluated using metrics such as provisioning latency and accumulated CPU time.

The design of a gateway that provisions resources to dead-line applications relying on information given by current resource management services may be complex. It basically depends upon scheduling decisions that are far from optimal.

So, when providers and brokers use conflicting policies, the number of migrations can be high [54]. In the same vein, even owners of individual resources will exhibit different usage patterns and implement different policies for the time they make their machines available [92].

This section reported about the resource provisioning and existing work on resource provisioning with QoS and without QoS. The next section would present the Grid scheduling methods in existing literature.

2.3 Grid Scheduling

Grid scheduling is an essential part of Grid resource management. Khateeb et al. define Grid scheduling as the process of making scheduling decisions involving allocation of jobs to resources over multiple administrative domains [65]. The Grid Scheduling Architecture Research Group (GSA-RG) of the Open Grid Forum (OGF) provides different Grid scheduling use case scenarios and also describes common usage patterns in [93]. Grid scheduling involves three main stages: resource information gathering, resource selection & provisioning and scheduling.

- Resource information gathering : In this phase, Grid scheduler collects information about the status of the available resources. This step is very important due to the heterogenous and dynamic nature of the Grid resources. Grid Information Service (GIS) provides the information such as CPU capacity, memory size, software availabilities and network bandwidth to Grid schedulers.

- Resource Selection and Provisioning : The performance of a resource for different application species is also necessary for making a feasible schedule. Application profiling is used to extract properties of application and analogical benchmarking provides a measure of how well a resource can perform a given type of job. Another alternative is using prediction and historical information and user specification. After the selection of resources from a set of resource pool, provisioning is performed.

- Scheduling: In this phase, the mapping of the job to the ingredient resource is performed along with the job execution.

In this section, an overview of resource scheduling techniques used in existing literature has been discussed.

Resource scheduling is a major concern to achieve high performance on computational Grids. It is basically a large-scale optimization problem due to the heterogenous and dynamic nature of the Grid resources.

Several heuristic optimization methods such as Genetic Algorithm (GA), Simulated Annealing (SA), Tabu Search (TS), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) among others, have been proposed for resource scheduling in computational Grids. Abraham et al. used nature's heuristics namely GA, SA and TS for scheduling of jobs on computational Grids. They have shown that GA performs better than TS and SA for scheduling of the jobs to exact resources but hybrid heuristic algorithms perform better than GA approach as it minimizes the time required for scheduling the job[94]. Liu et al. [95] considered PSO to design a job scheduling algorithm in computational Grids. They have used the concept of a fuzzy matrix to design the scheduling algorithm. These scheduling methods try to minimize the execution time /makespan of the applications and as such are suitable for Grids. However, in Grids, there is another important parameter other than execution time, i.e., cost. Meta-heuristic approach has been used in all these methods but the hyper-heuristic approach would have performed better in comparison to the meta-heuristic.

Garg et al. proposed a model for meta-scheduling on utility Grids using linear programming and a genetic algorithm. This model minimizes the cost for scheduling of an independent task. They have considered multiple and concurrent users which are competing for the resources in a meta-scheduling environment so as to minimize their cost [96]. Garg et al. [97] proposed three novel heuristics for parallel applications on utility Grids. They evaluated the sensitivity of the proposed heuristics on the basis of changes in user's preference, application's execution time and resource's pricing.

Brun et al. [98] compared eleven static heuristics for mapping a class of independent tasks on heterogenous computing systems. They used a simulation model for comparing these static heuristics which returned the best optimum results. Xhafa and Abraham [10] surveyed the computational model and heuristic methods for Grid scheduling problems. They only surveyed the existing approaches. Gaoa et al. [99] developed two algorithms that used the predictive models to schedule jobs at both the system-level and application-level. In application-level scheduling, GA is used to minimize the average completion time of jobs through optimal job allocation on each node.

Golconda et al. did the comparison of five heuristics, namely QSMTS-IP, min-min, GA, least slack first and suffrage. The comparative study provides a fair basis for the comparison of these heuristics [100]. Kim et al. [101] proposed a novel GA based approach that automatically decomposes data into communication and computation resources for scheduling of a divisible data intensive application. Konugurthi et al. [102] proposed a heuristic based GA for an optimal mapping of application/jobs to suitable resources.

Kondo et al. [103] designed three general techniques for resource selection: resource prioritization, resource exclusion and task duplication for scheduling of task parallel applications for rapid turnaround on enterprise desktop Grids. They have used techniques to instantiate several scheduling heuristics and investigated three approaches for task replication namely proactive, reactive and hybrid methods. The hybrid replication heuristic based on a probabilistic model of task completion did not perform any better than the best reactive heuristic. Jun et al. proposed a task scheduling heuristic algorithm based on linear programming for tree based Grid computing platforms. In this algorithm, they considered the computing power and bandwidth for each node in the model and assigned task for each node in an integrated manner. Then they used SimGrid to simulate and analyze the algorithm [104]. Chauhan et al. proposed QoS guided weighted mean time-min and QoS guided weighted mean time min-min max-min selective for QoS based Grid task scheduling. They have used the concept of historical information about the execution time of jobs to predict the performance of the algorithm [105].

Roy et al. [106] have discussed the resource brokering strategies within the multi-agent framework. These strategies help in finding out an optimal allocation of resources for executing multiple concurrent jobs in a Grid environment. Somasundaram et al. have proposed the CARE Resource Broker (CRB) [107] that addresses scheduling scenarios like unavailability of resources and the required execution environment for the resource providers. CRB is a Grid meta-scheduling framework that supports virtualization technology only.

Cameron et al.[108] have studied and analyzed the effects of various job scheduling and data replication strategies and compared them in a variety of Grid scenarios using several performance metrics. They used the Grid simulator OptorSim, and based their simulations on a world-wide Grid testbed for data intensive high energy physics experiments.

Saleh et al.[109] introduced adaptive Grid scheduling strategy that employs mobile agent technology in both monitoring and rescheduling processes. The main use of integrating mobile agent technology with computational Grids is to minimize the network traffic during the Grid resource discovery and monitoring processes. They only considered the scheduling and rescheduling of computational tasks neglecting QoS parameters like cost and security. This approach does not take QoS requirements such as time into consideration.

Sanjay et al. have developed three strategies for deciding when and where to reschedule parallel applications that execute on multi-cluster Grids [110]. They only focused on the rescheduling instead of scheduling.

Desprez et al.[111] designed scheduling and replication algorithm that computes the mapping of data and computational requests at the same time. This approach used knowledge of the database usage scheme and of the target platform. Their main focus was to manage data and their replication. They mainly used the greedy algorithm to solve the mapping problem. The idea behind this algorithm is to try to map data that needs the maximum computational power to the server that has the maximum computation capacity. Pugliese et al. [112] analyzed the requirements of Grid resource management and provided a classification of schedulers. After that, they defined an extensible formal model for Grid scheduling activities. They designed only scheduling model instead of a resource scheduling algorithm to schedule jobs on the available resources. They did not consider the cost, makespan, security and reliability for independent job scheduling in the Grid environment.

Song et al. designed a new fuzzy-logic trust model for distributed trust aggregation through fuzzification and integration of security attributes [113]. The trust model aggregates many reputation attributes and measurable self-defense capabilities into scalar quantities, which can be easily applied to quantify the trust index of Grid resource sites. Their security-binding scheme scales well with increasing user jobs and Grid sites but they did not design scheduling algorithm for application schema. Additionally, fitness function of the genetic algorithm is dependent on the makespan of the solution while ignoring the other scheduling criteria like cost and security.

Cowling et al. stated that hyper-heuristic is a good approach to solve various problems such as personal scheduling, timetabling, nurse scheduling and resource scheduling. It is not problem specific like the meta-heuristic approach and it can be efficiently applied to an optimization problem [114]. Gonzalez et al. [115] used ad hoc (immediate and batch mode) scheduling methods to design a hyper-heuristic approach for scheduling of jobs on the Grid nodes according to the job and Grid characteristics. Bhanu et al. [116] designed a scheduling model for resource scheduling using heuristic methods. They have used Longest Job Faster Resource (LJFR) heuristic and Shortest Job Faster Resource (SJFR) heuristic method for resource scheduling.

The resource scheduling problem in Grid becomes more challenging as it is important to achieve not only the promising potentials of tremendous distributed resources, but also effective and efficient scheduling algorithms. Hence, a lot of algorithms have been developed for scheduling jobs in a computational Grid with an aim to minimize the job completion time [117].

Different Grid scheduling approaches have been investigated and applied to different Grid scenarios and requirements. Some of them have tried to schedule the jobs securely but there is not even a single approach that can be found in the literature that covers resources provisioning based scheduling. Kyriaki et al. have considered multi-criteria job scheduling using accelerated genetic algorithm [117]. They have considered security constraint for scheduling of jobs but resource provisioning based scheduling has not been considered. Kolodziej et al. [118][119] have presented an approach for independent task scheduling with security requirements in Grid computing environment. They have developed a scheduling model that enables the aggregation of task abortion and security requirements. They have used game-theoretic and a GA based approach for optimizing the makespan and flowtime. Menasce et al. stated that the main problem in Grid environment is the selection of services and service providers in an appropriate way so as to achieve global SLA with minimum cost [120].

The next section discusses scheduling in the existing Grid middleware.

2.3.1 Scheduling in Grid Middleware

Grid middleware is a layer between user's application and resources. The main aim of the Grid middleware is to find convenient places for applications to run, optimize the use of resources, organize the efficient access to data, deal with authentication to the different sites that are used, run the job & monitor progress, recover from problems and transfer the result back to the scientists. In this section, an overview of Grid middleware and their scheduling techniques have been discussed. The taxonomy followed is based on the scheduling models of gLite [121], UNICORE [122], Globus [123] and Legion [124]. Based on this taxonomy, an analysis of these middleware has also been done.

a) gLite middleware: gLite is the next generation middleware for Grid computing. It was born from the collaborative effort of more than 80 people in 12 different academic and industrial research centers as a part of the EGEE Project [121]. gLite provides a framework for building Grid applications those tapping into the power of distributed computing and storage resources across the internet. gLite is a lightweight middleware for Grid computing and it provides scalability, performance, interoperability and modularity.

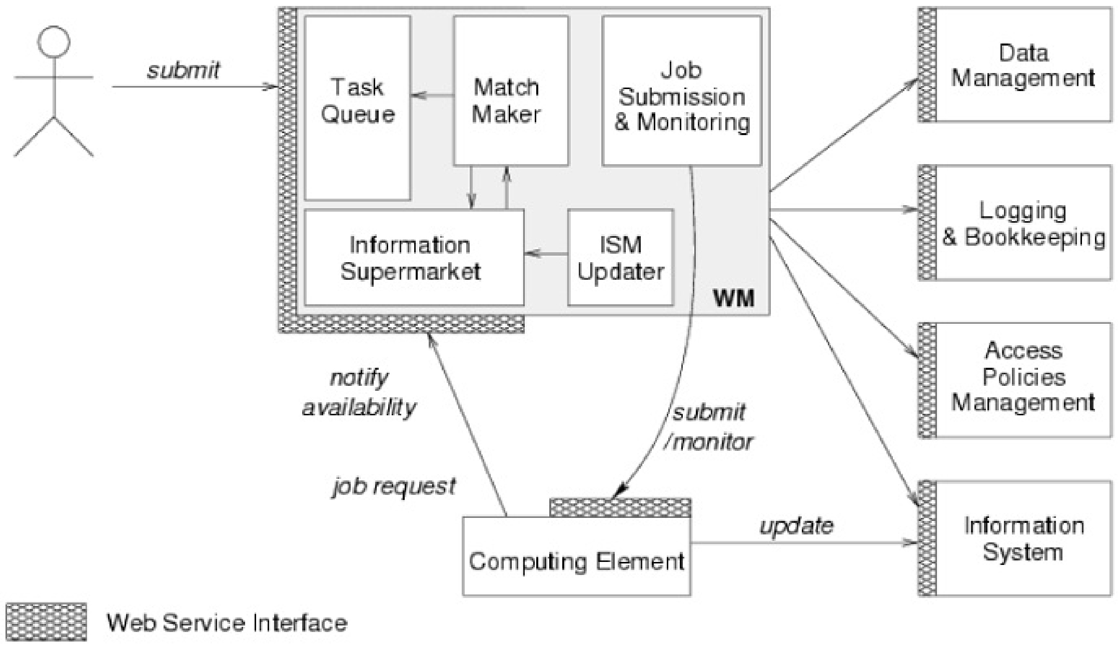

Figure 2.1: Job management services in gLite [125]

Scheduling in glite: gLite middleware achieves efficiently and reliably the scheduling of computational tasks on the available infrastructure. gLite consists of six principle components: Computing Element (CE), Workload Management (WM), Storage Element (SE), Catalog, Information and Monitoring and Security.

CE is a gateway to the local batch system & cluster. It provides the service that represents the computing resource that is responsible for the job management (submission, control, etc.). The CE may be used by a generic client, an end-user interacting directly with the computing element, or the workload manager, which submits a given job to an appropriate CE found by a matchmaking process [125].

Workload management acts as a resource broker in gLite middleware as shown in Figure 2.1. It allows user to submit the job and performs all tasks that are required to execute without exposing the user to the complexity of the Grid [126]. Workload management is primary job execution interface for users to find the best location for a job, considering job requirements and available resources (CPUs, files) and it gets resource information from

InfoSystem and File Catalogs. It is a set of middleware components that is responsible for the distribution and management of jobs across Grid resources [127]. In this, "matchmaking" the best available resource is assigned and "Logging & Bookeeping" keep track of job execution in term of events (Submitted, Running, Done,...).

Storage element is the gateway to a local storage (disk, tape) and a Gridftp server. Catalog remembers locations of files and it only deals with their locations.

In gLite, the task of information accumulation about the resource usage is done by users or group of users (VOs). Information on Grid services/resources needs sensors (resource metering, metering abstraction layer, usage records). Records are collected by the accounting system (Queries: Users, Groups, Resource). A system called Virtual Organization Membership Service (VOMS) is used to manage information about the roles and privileges of user with in a VO. This information is presented to service via an extension to the proxy [125].

b) UNICORE: In 1997, the development of the UNIform Interface to COmputer REsources (UNICORE) system was initiated to enable German super computer centers to provide their users with a seamless, secure and intuitive access to their heterogeneous computing resources [122][128].

Scheduling in UNICORE: The UNICORE architecture is based on three tier architecture: user tier, server tier and target tier. The user tier consists of the graphical user interface and offers the functions to prepare and control the UNICORE jobs and to set up and maintain the user's security environment [129]. A UNICORE (sub-) job is executed on behalf of a UNICORE user account. The interaction between a UNICORE client and its server is transaction based and usually asynchronous. So, the job is submitted to a UNICORE server and only the receipt of the job is acknowledged. The client does not wait for the completion of the job. Jobs may be submitted synchronously but the UNICORE server is free to convert this to asynchronous if the execution takes too long. The actual abstract job which is sent from the client to a Vsite is constructed in the client as an Abstract Job Object (AJO) [130]. The Java AJO class library is the basis for the modeling of UNICORE jobs and for the protocol between the UNICORE clients and the servers together with the abstract job specification generated from the user input [129][131]. The AJO is a key component in the architecture. Most

Grid scheduling actions within a UNICORE environment are currently carried out manually [132].

c) Globus: The Globus Alliance produced widely-used Grid middleware known as the Globus Toolkit. Globus is a software toolkit addressing key technical problems in the development of Grid enabled tools, services, and applications [123]. Globus is an open community project based on Apache Jakarta model. The Globus toolkit is a community-based, open-architecture, open-source set of services and software libraries that supports Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, fault detection, and portability.

   Scheduling in Globus: Grid Resource Allocation Management (GRAM) is a software component of the Globus Toolkit that can locate, submit, monitor, and cancel jobs on Grid computing resources. It handles placement, provisioning and lifetime management of jobs. GRAM is a unifying remote interface for remote job submission and resource management. In globus, Grid job goals provide an environment for the job, stage files to/ from environment, causes, monitors the job execution, send job state change notifications and streams a job's stdout/err during execution. In GT4, remote job execution is carried out by the GRAM service, which is instrumental for providing a web service interface for submitting requests to execute jobs, defined in a job description language. It also monitors and controls the resulting job executions. GT4 includes two different GRAM services: the 'pre-WS GRAM', introduced in GT2, and the newer Web Services -based 'WS GRAM', introduced in GT4. When a job is submitted to the Globus, GRAM sends it to different distributed resources. When the user client sends the job request from the local user, the request is forwarded by this meta-scheduler to a Grid manager who in turn queries the Resource Manager (RM) for resources. RM stores information about local and remote resources. If available resources are found on a local site, job request is forwarded to the local scheduler. If the resources are not available locally, job request is sent to a remote site 's globus gate-keeper , which in turn forks the job to the job manager and is finally submitted to the remote scheduler for execution [133] [134]. The job details are specified through the Globus Resource Specification Language (RSL), which is the main part of the GRAM [134].

d) Legion: Legion is an object-based, meta-systems software project at the

University of Virginia [124]. It is a middleware system that combines very large number of independently administrated heterogeneous hosts, storage systems, databases legacy codes and user distributed objects distributed over wide area network into a single coherent computing platform [134]. The main goal of the legion middleware is to promote the principle design of the distributed systems software by providing standard object representation for file systems, data systems and processors, etc.

Scheduling in Legion: Scheduling in legion is not of a dictatorial nature;

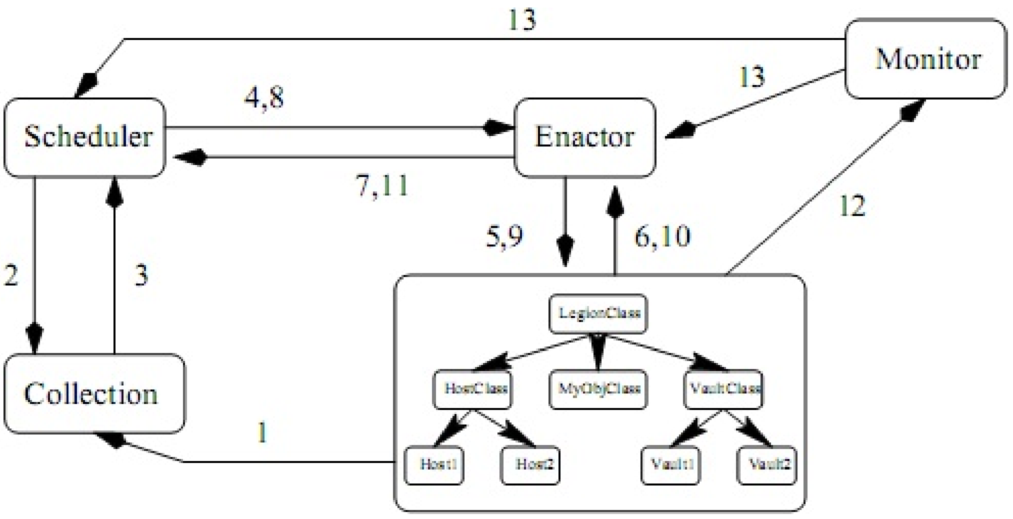

Figure 2.2: The use of Resource Management architecture in Legion [135]

request is made to the resource guardians who have the final authority [135]. Figure 2.2 shows that the user defined schedulers will directly interact with the infrastructure. Hosts and vault are the basic resources of the model. The collection is an information database and the enactor is the scheduler implementor of the resource management model. At first, the collection is populated with information describing the resources. Then the scheduler queries the collection and after getting the result of the query, computing of objects to resources is done. In case of application specific scheduler, there should be an appropriate knowledge of the application and then the mapping of application's classes is done. Scheduler passes the mapping to enactor, who invokes the methods based on hosts and vaults and gets the information of resource reservation from the mapping. After getting the information of resource reservation, enactor consults with the scheduler for confirmation, then the enactor instantiates objects through member function calls on the appropriate class objects. The class object gives the success or failure result which is informed to the scheduler via enactor. If during migration, there is a need for the object to be migrated then this is notified to the scheduler and enactor and they reschedule the object.

e) Aneka: Aneka can be used as Grid middleware but recently aneka has also become market oriented Cloud development and management platform with rapid application development and workload distribution capabilities [136]. Aneka provides some advantages over Grid and cluster based workload distributions like provisioning based QoS/ SLA, support of multiple programming and application development, simultaneous support of multiple runtime environments. Aneka can be used in drug design, medical imaging, moduler & quantum mechanics and genomic research.
Scheduling in Aneka: In aneka, the scheduling services schedule work unit to execution nodes based on availability, capability and QoS requirements and the particular scheduling algorithms used. Pluggable scheduling algorithms, advance reservation and automatic/custom resubmission of failed task features are provided by Aneka scheduling services [137].

Table 2.1: Comparison of Grid Middleware

Middleware/ Features	**Advance reservation**	**QoS**	**Resource Provisioning**	**Scheduling Organization**	**Scheduling Policy**
gLite	Yes	Yes	Yes	Distributed	User Centric
Unicore	Yes	Yes	No	Distributed scheduler	User centric
Globus	Yes	Soft QoS	Yes	Hierarchical Scheduler	Ad-hoc extensible policy
Legion	Yes	Soft QoS	Yes	Hierarchical Scheduler	Ad-hoc extensible policy
Aneka	Yes	Yes	Yes	Decentralized & flexible	System centric

Table 2.1 describes the comparison between the above discussed middleware w.r.t to the features that they provide.

Next section discusses about a taxonomy of Grid schedulers.

2.3.2 A Taxonomy of Grid Schedulers

Grid scheduler works for the mapping of jobs to exact resources according to the requirements of the users. A Grid scheduler must make scheduling decisions in the best manner and submit the job to the selected resource. Grid scheduler is different from local scheduler [62]. Local scheduler only manages a single site or a cluster and usually owns the resources but this is not the case with the Grid scheduler. Grid scheduler is the key component of a computational Grid as it is responsible to optimize the use of Grid resources. A number of schedulers for Grid computing systems have been developed. In this section, following schedulers have been surveyed after considering their important features.

CONDOR-G: Condor [138] is a high throughput scheduler. It runs the job on single administrative domain but Globus runs the job on multiple administrative domains. By combining the strength of both, Condor- G is obtained. In Condor-G, Condor pool is divided into two parts: job management and condor software. Condor-G is the job management part of the Condor and Condor software does the resource management [139]. It uses the Globus toolkit to start a job on a remote machine instead of using condor-developed protocol. It uses globus protocols like Grid Resource Allocation & Management (GRAM), Global Access to Secondary Storage (GASS), Replica Location Service (RSL) and Grid Security Infrastructure (GSI) [140]. Condor-G also fulfills the requirements of the job' and tells the status of each job. Condor- G scheduler allows submitting the jobs into a queue, having log details of the life cycle of jobs and handling all input and output files. It provides a ”window to the Grid” for users to both access resources and manage the jobs running on remote resources. The advantages of Condor- G is that it provides the facility of credential management, fault-tolerance and full -featured queuing services but the disadvantage is that it is platform dependent.

NIMROD-G: Nimrod-G is a resource broker and it performs resource management and scheduling of parameters sweep and task-farming applications on world-wide Grid resources [141] [142]. Nimrod-G uses globus and legion as Grid middleware for resource discovery/information and uses either network directory or object model based data organization. Load balancing facility is also provided by Nimrod-G by using rescheduling.

GridWay Scheduler: GridWay is a meta-scheduler (A meta-scheduler uses local schedulers of the particular systems. Thus, meta-schedulers coordinate local schedulers to compute an overall schedule [10]) which enables large-scale, reliable and efficient sharing of computing resources such as clusters, supercomputers and stand alone servers [143][144]. PBS, LSF, SUN GRID ENGINE, CONDOR are supported by GridWay within a single organization or scattered across several administrative domains. As local resource management system, it uses globus WS, Pre WS, LCG, CREAM, ARC, etc. as a middleware for Grid applications [140]. First of all, client application agent uses client API to communicate with the request manager in order to submit the job. After this, client may also request the request manager to control the operation of job like start, resume and rescheduling of job. Then the dispatch manager submits pending jobs at each scheduling interval and reschedules the job. It also invokes the resource selector who gets information from monitoring and discovering services and short lists candidate hosts. The dispatch manager submits pending jobs by calling a submission manager, and also checks that wether the migration of rescheduled jobs is worthwhile or not. If this is the case, the dispatch manager triggers a migration event along with the new selected resource to the job submission manager, who manages the job migration [145]. The submission manager is responsible for the execution of the job during its life time, i.e. until it is done or stopped. It is initially invoked by the dispatch manager along with the first selected host, and is also responsible for performing job migration to a new resource [145]. The performance monitor periodically wakes up at each monitoring interval. It requests rescheduling actions to detect "better" resources when performance slowdown is detected [145][146].

SUN GRID ENGINE: Sun Grid Engine (SGE) is an open source and provides dynamic resource management services like load balancing, maximizes resource utilization, transparent job submission & machine selection, monitoring and accounting [22]. It acts as a resource broker for globus and takes scheduling decisions to select a remote site. Sun Grid engine broker provides services like guaranteeing required resources, full control over resource utilization, fair usage of resources that are otherwise shared and implementation of management policies. SGE/Broker submits and tracks jobs to remote systems using Globus services.

PBS: Portable Batch System (PBS) is a package which was designed and written by the Numerical Aerodynamic Simulation Complex, NASA, in 1994. PBS is a successor to Network Queuing System (NQS) and has addressed many of the deficiencies of NQS. It was designed to provide additional controls over the initiating, or scheduling, of execution of batch jobs [147] [148]. OpenPBS is the original version of the Portable Batch System (PBS) , a queuing system developed for NASA in the early 1990s [149]. OpenPBS operates on networked multi-platform UNIX environments. PBS Pro is used for job scheduling. In 1999, PBSpro and Globus were interoperable. Since 2000, it is a commercial product. It consists of a Server Daemon (1 server per cluster) and a Scheduler Daemon (1 scheduler for unique cluster) - it is also possible to include an external scheduler. The Machine Oriented Mini-Server (MOM) Daemon runs on each execution node and is responsible for accounting and job processing. There are web-interfaces to CAE computations. It integrates Unix, Linux and there is a Grid front-end to Globus [150]. The future PBSpro offers fault tolerance and reliability, suspend/resume, checkpoint/restart, SMP cluster, floating licenses and supports system specific features (e.g. PSSP of IBM SP).

Gridbus Grid Service Broker: The Gridbus [151] project is an open source, multi-institutional project led by the GRIDS lab at the university of Melbourne. The Gridbus broker is designed for data Grid and computational Grid applications. The Gridbus broker follows a Service-Oriented Architecture (SOA) and is designed on object-oriented principles with a focus on the idea of promoting simplicity, modularity, reusability, extensibility and flexibility [152]. Gridbus emphasizes the end-to-end quality of services driven by computational economy at various levels - clusters, peer-to-peer (P2P) networks, and the Grid - for the management of distributed computational, data, and application services [134]. The Gridbus resource broker provides an abstraction to the complexity of Grids by ensuring transparent access to computational and data resources for executing a job on a Grid. It uses user requirements to create a set of jobs, discover resources, schedule, execute and monitor the jobs and retrieve their output once they are finished [153][134].

LSF: Load Sharing Facility (LSF) by Platform Computing is the most powerful workload manager for demanding distributed and mission-critical high performance computing environments [154]. It provides a complete set of workload management capabilities and is designed to work together to address high performance computing needs. Platform LSF includes a comprehensive set of intelligent, policy-driven scheduling features, enabling to fully utilize compute infrastructure resources. It is highly scalable and allows the user to schedule complex workloads[154]. With the best support in the HPC industry, Platform LSF provides the most complete HPC datacenter solution for workload management.

A comparison of Grid schedulers which classifies schedulers by characterizing different attributes is summarized in Table 2.2. The comparison focuses on their architecture, Grid type, language, platform, service and QoS parameters.

This section reported about the taxonomy of Grid schedulers and the next section would present the Grid scheduling algorithms.

Table 2.2: Comparative Analysis of Grid Schedulers

Scheduler/ Features	**Architecture**	**Grid Type**	**Language**	**Platform**	**Services**	**QoS parameters**
Condor-G	Independent Cluster	Computational Grid	Java	Linux, Solaris, Digital Unix, IRIX	Job management, resource selection, security, Fault Tolerance	Reliability
Nimrod-G	Resource Broker	Computational Grid, Service Grid	Parametric, declarative language	Platform Independence	Resource discovery,selection, scheduling, transparent execution of user jobs on remote resources	According to user requirements, Cost, Availability of resources, Hard and soft QoS
GridWay	Resource Broker	Partner, Supply Chain Grid, Enterprise Grid	C,java,perl, ruby, Python binding	Support existing remote platform	Advance scheduling, dynamic discovery, selection, opportunistic migration, performance slowdown detection, support for self-adaptive, fault detection, recovery	Flexiblity, Extensibility, Efficiency, Reliability
Sun Grid Engine	Independent Cluster	Data Grid, Computational Grid	DRMAA API,C, C++, Fortran, Netbeans	Multiple Cross Platforms	Distributed job scheduling,Resource balance,time management, authentication, authorization, job monitoring, Fault tolerance	Flexible administration, advance reservation, scalability
PBS	Independent Cluster	Computational Grid	Torque	Runs on most Unix-like systems: e.g. Linux, Irix, Unicos, HPUX, IA64, etc.	Preemptive job scheduling,Scheduler backfilling, Improved fault-tolerance, Desktop Cycle Harvesting	Advance reservation, flexibility, reliability, cost effective, scalability, serviceability
Gridbus Grid Service Broker	Resource Broker	Data Grid, Computational Grid	Java	Windows, Linux	Resource discovery, job scheduling, monitoring of jobs	Flexibility, Simplicity, HPC Solutions, Extensibility
LSF	Independent Cluster	Data Grid, Computational Grid, Enterprize Grid	C, Phython	Windows, Linux	Load sharing, Job scheduling	Flexibility, Visibility, Highly scalable, Advance Reservation

2.3.3 Grid Scheduling Algorithms

In literature, Grid scheduling algorithms have been discussed from different perspectives, such as static vs. dynamic policies, objective functions, application models, adaption QoS constraints, strategies dealing with dynamic behavior of resources and so on [26]. Further, scheduling can be done as static vs. dynamic, distributed vs. centralized and co-operative vs. non- co operative. Existing techniques and algorithms for scheduling of Grid resources are:

i. **Local versus Global**: The local scheduling discipline determines how the processes resident on a single CPU are allocated and executed where as a global scheduling policy uses information about the system to allocate processes to multiple processors so as to optimize a system-wide performance objective. Grid scheduling should be done as global scheduling [26].

ii. **Static versus Dynamic**: In Grid, both static and dynamic types of scheduling have been adopted. In static scheduling, information regarding all the resources in the Grid as well as all the tasks in an application are assumed to be available at the time of scheduling but in the case of dynamic scheduling, the basic idea is to perform task allocation on the fly as the application executes.

iii. **Centralized versus Decentralized**: In Grid scheduling, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In centralized scheduling, Grid schedular has more control on the resources and in this case efficient scheduler can be designed. Centralized Grid scheduling algorithm can be easily implemented but it suffers from the lack of scalability, fault tolerance, etc. Therefore, centralized scheduling is not useful for large-scale Grids. In decentralized scheduling, Grid schedulers have no centralized control over the resources. In this case, local schedulers play an important role in scheduling and also manage and monitor the status of the resources [10].

iv. **Co-operative versus Non-cooperative**: In co-operative scheduling, each Grid scheduler carries out its own scheduling tasks, but all schedulers are working toward a common system-wide goal. Scheduling is done through the co-operation of Grid users, rules and policies. In the non-cooperative case, each scheduler acts alone as an autonomous entity and arrives at decisions regarding their own optimum objects independent of the effects of the decision on the rest of the system [26].

v. **Approximation versus Heuristics**: The approximate algorithms use formal computational models, but instead of searching the entire solution space for an optimal solution, they are satisfied when a solution that is sufficiently "good" is found. Heuristic algorithms are more adaptive to the Grid scenarios where both resources and applications are highly diverse and dynamic, so heuristics are considerably a de-facto approach for solving Grid scheduling problems [155].

The next section presents a detailed study of Grid scheduling heuristic approaches.

2.4 Grid Scheduling Heuristics

The mapping of jobs to appropriate resources for execution of application in heterogeneous environments like Grid computing is called an NP-complete problem [98]. NP-complete problems are often solved using heuristic methods. Heuristic approaches can be applied to Grid scheduling problems because Grid scheduling has various important issues that needs to be addressed such as heterogeneity of the resources, dynamic and autonomous nature of Grid resources and finally resource providers and resource consumers have different policies for the execution of their applications. Existing scheduling heuristics as given below are discussed and compared in [98].

2.4.1 Greedy Heuristic approaches

Greedy algorithms are intuitive heuristics in which greedy choices are made to achieve a certain goal [156]. Greedy heuristics are constructive heuristics since they construct feasible solutions for optimization problems from scratch by making the most favorable choice in each step of construction. By adding an element to the (partial) solution which promises to deliver the highest gain, the heuristic acts as a greedy constructor.

Opportunistic Load Balancing: Opportunistic Load Balancing (OLB) assigns each task, in arbitrary order, to the machine that is expected to be available next. This is regardless of the task's expected execution time on that machine [157]. The main aim of the OLB is to keep all machines as busy as possible. One advantage of OLB is its simplicity, but as OLB does not consider expected task execution times, the mappings it finds can result in very poor makespans.

Minimum Execution Time: In contrast to OLB, Minimum Execution Time (MET) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task. This is regardless of that machine's availability [98]. The motivation behind MET is to give each task to its best machine. It does not consider the current load on a resource so will often cause load imbalance between the processors.

Minimum Completion Time: Minimum Completion Time (MCT) assigns each task, in arbitrary order, to the machine with the minimum expected completion time for that task [157]. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The intuition behind MCT is to combine the benefits of OLB and MET, while avoiding the circumstances in which OLB and MET perform poorly.

Min-min: The Min-min heuristic considers all the unmapped tasks with the known set of minimum completion times. Next, the task with the overall minimum completion time is selected and assigned to the corresponding machine (hence the name Min-min) [157] [158]. Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min-min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could.

Max-min: The Max-min heuristic is very similar to Min-min. The Max-min heuristic also begins with the set of all unmapped tasks with the known set of minimum completion times. Next, the task with the overall maximum completion time is selected and assigned to the corresponding machine (hence the name Max-min). Intuitively, Max-min attempts to minimize the penalties incurred from performing tasks with longer execution times [157] [158]. Assume, for example, that the metatask being mapped has many tasks with very short execution times and one task with a very long execution time. Mapping the task with the longer execution time to its best machine first allows this task to be executed concurrently with the remaining tasks (with shorter execution times). For this case, this would be a better mapping than a min-min mapping, where all the shorter tasks would execute first, and then the longer running task would execute while several machines sit idle. Thus, in cases similar to this example, the Max-min heuristic may give a mapping with a more balanced load across machines and a better makespan.

Duplex:The Duplex heuristic is literally a combination of the Min-min and Max-min heuristics. The Duplex heuristic performs both the Min-min and Max-min heuristics and then uses the better solution. Duplex can be performed to exploit the conditions in which either Min-min or Max-min performs well, with negligible overhead [98].

2.4.2 Local Search based Heuristic approaches

Local search heuristic approaches is a family of methods that explore the solution space by starting at an initial solution, and constructs a path in solution space during the search process [10]. Local search heuristic approaches improve solutions through neighborhood search. The main objective of this local search based heuristic approach is to gain feasibility as soon as possible [159]. They have been applied successfully to many industrial problems and performance of local search based heuristic approaches depend on construction of neighborhood.

Tabu Search: Tabu Search (TS) is a high level heuristic procedure for solving optimization problems and was proposed by Glover in 1986. It is a meta-heuristic that guides a local search procedure to explore the solution space beyond local optimality [160]. TS avoids entrapment in local minima and continues the search to give a near optimal final solution. It is very general and conceptually much simpler than other meta heuristic algorithms such as genetic algorithm, simulated annealing and ant colony optimization algorithms. TS is very easy to implement and does not require special memory space. TS takes short searching time to solve combinatorial optimization problems. It uses specific set of constraints, known as tabu conditions, in order to avoid blind search. TS has disadvantages like it often gets locked in looping from one local optimum to another and it has low global search capability.

Hill Climbing: Hill Climbing (HC) is a graph search algorithm where the current path is extended with a successor node which is closer to the solution than the end of the current path. It is logical and beneficial especially in situations where the search space is of simple nature with no more than a single maxima or minima [10]. HC is a local search heuristic technique and it is more simpler and straight forward in comparison to other heuristics. The main disadvantage in case of hill climbing is that, the solution is better than all of its neighbors, but it is not better than some other states far away. There is a flat area of the search space in which all the neighboring states have a same value. HC is a local method and it moves in many directions at a time.

Simulated Annealing: Simulated Annealing (SA) heuristic approach was proposed by kirkpatrick et al. in 1983. The simulated annealing process consists of first melting the system being optimized at high effective temperature, then lowering the temperature by slow stages until the system freezes and no further changes occur [161]. It is an iterative technique that considers only one possible solution (mapping) for each meta task at a time [162]. This solution uses the same representation as the chromosome for the genetic algorithm. The initial implementation of SA was evaluated and then it was modified and refined to give a better final version in [98]. It uses a procedure that probabilistically allows poorer solutions to be accepted in an attempt to obtain a better search of the solution space.

SA has many advantages: (i) It is guaranteed to converge in asymptotic time. (ii) It can deal with arbitrary systems and cost functions. (iii) SA statically guaranties to find an optimal solution and it is relatively easy to code, even for complex problems. (iv) It is robust heuristic to implement and has an ability to provide reasonably good solutions for many combinatorial problems. The main disadvantage of SA is that there is a need for a great deal of computer time for many runs and carefully chosen turnable parameters. It has a difficulty in defining a good cooling schedule which is important both in single and multi objective optimization. In case of SA, if there is a repeated annealing with 1/logk then the scheduling is very slow, especially if the cost function is expensive to compute.

2.4.3 Population based Heuristic approaches

Population-based heuristic is a large family of methods which are highly efficient for solving combinatorial optimization problems. However, when the objective is to find feasible solutions of good quality in short execution times, as in the case of Grid scheduling, the inherent mechanisms of these methods can be exploited to increase the convergence of the method [10].

Genetic Algorithms: Genetic Algorithm (GA) was proposed by hollland et al [163]. GA is a famous stochastic optimization algorithm which uses biologically inspired techniques such as genetic inheritance, natural selection, mutation, and sexual reproduction (recombination, or crossover) [164]. It is a useful heuristic to find a near optimal solution in large search spaces [162]. In GA, a point in search space is represented by a set of parameters and these parameters are known as genes and a set of genes is known as string or a chromosome. A fitness function must be devised for each problem to be solved. Each chromosome is assigned a fitness value that indicates how closely it satisfies the desired objective. Given a particular chromosome, the fitness function returns a single numerical fitness or figure of merit, which will determine the ability of the individual, which that chromosome represents [94]. A set of chromosomes is called population. Reproduction is another critical attribute of GAs where two individuals selected from the population are allowed to mate to produce offspring, which will comprise the next generation. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. Mutation provides a small amount of random search, and helps ensure that no point in the search space has a zero probability of being examined. If the GA has been correctly implemented, the population will evolve over successive generations so that the fitness of the best and the average individual in each generation increases towards the global optimum. The genetic algorithms have been found to be very powerful in finding out a global minima [165][98]. Genetic algorithms have been applied to many classification and performance tuning applications in the domain of knowledge discovery in databases [164] [166].

The important advantages of GA are: (i) Analytical knowledge is not required. (ii) It is easy to parallelize and no derivatives are required (iii) GA works on a wide range of problems and has better global capability. The disadvantages of genetic algorithm: (i) It requires much more evolution functions than linearized methods. (ii) There is no guaranty of convergence of local minima. (iii) It converges to local optima or arbitrary point rather than the global optima of the problem. (iv) GA has a slow convergence rate and premature convergence and (v) it can not use the feed back of a system.

Memetic Algorithm: Memetic Algorithm (MA) is an extension of genetic algorithm. Memetic algorithms are evolutionary algorithms that can be applied on a local search process to refine solutions for hard problems. It is the subject of intense scientific research and have been successfully applied to a multitude of real-world problems ranging from the construction of optimal university exam timetables, to the prediction of protein structures and the optimal design of spacecraft trajectories [167]. MA can handle complex objective functions. It combines the advantages of local search and genetic algorithm for optimization problems. MA can also be used for global search. It is based on a genetic algorithm and extended by a search technique to further improve individual fitness that may keep with population diversity and reduce the likelihood premature convergence. MA requires a considerable amount of time and memory needed for the improvement of their performances. MA can be used only in non-linear continuous multi-objective combinatorial optimization problems.

Ant Colony Optimization: Ant Colony Optimization (ACO) was proposed by

Marco Dorigo in 1992 [168]. The real power of ants resides in their colony brain. The self-organization of those individuals is very similar to the organization found in brain-like structures. Like neurons, ants use mainly chemical agents to communicate. One ant releases a molecule of pheromone that will influence the behavior of other ants [169]. Ant algorithms are non-deterministic and rely on heuristics to approximate to a sub-optimal solution in cases where the number of combinations is extremely huge and is impossible to calculate using a deterministic algorithm [170]. Ant algorithms are often compared with other evolutionary approaches such as Genetic Algorithms, Evolutionary Programming and Simulated Annealing.

Following advantages have been identified in [171]: (i) It is versatile and can be applied to similar versions of the same problem; for example, there is a straight forward extension from the Traveling Salesman Problem (TSP) to the Asymmetric Traveling Salesman Problem (ATSP). (ii) It is robust and can be applied with only minimal changes to other combinatorial optimization problems such as the Quadratic Assignment Problem (QAP) and the Job-shop Scheduling Problem (JSP). (iii) It can be used for static and dynamic combinatorial optimization problems. (iv) ACO convergence is guaranteed and it used for solving constrained discrete problems. (v) ACO has the powerful feedback capability which can increase the speed of evolution of algorithm to make algorithm convergence possible in the end. When the graph changes dynamically, the ant colony algorithm can run continuously and adapt to changes in real time. Some of the disadvantages are: (i) ACO's convergence rate is slow in comparison to other heuristics. (ii) ACO performs poorly for larger city in TSP. (iii) In ACO, there is no centralized control to guide and provide good solutions. (iv) ACO can be applied to only discrete problems and theoretical analysis in ACO is difficult.

Particle Swarm Optimization: Particle Swarm Optimization (PSO) is a method for performing numerical optimization without explicit knowledge of the gradient of the problem to be optimized. PSO is one of the latest evolutionary optimization techniques inspired by nature and was introduced in 1995 by Kennedy and Elberhart [172]. It simulates the process of a swarm of birds preying. It has a better ability of global searching and has been successfully applied to many areas. A flock or swarm of particles is randomly generated. Initially, each particle position represents a possible solution point in the problem space. The fitness value of each particle is evaluated by the objective function to be optimized. Each particle remembers the coordinates of the best solution (gbest) achieved so far. The coordinates of the current global best (pbest) are also stored. The key advantages of this approach are: (i) PSO is a robust stochastic optimization based on the movement and intelligence of swarms. There is no selection and crossover parameter like genetic algorithm. (ii) PSO is easy to implement, few parameters to adjust, computationally efficient etc. (iii) PSO is efficient for global search algorithm. Some of the disadvantages are: (i) PSO has week local search. (ii) PSO has slow convergence rate in refined search strategy.

Bacterial Foraging Optimization: Bacterial Foraging Optimization (BFO) algorithm was proposed by Passino [173]. It is population based numerical optimization algorithm based on foraging behavior of Escherichia coli bacteria. In the foraging theory, the objective of the animal is to search and obtain nutrients in a fashion that energy intake per unit time (E/T) is maximized. Foraging is a process in which a group of bacteria moves in search of food in a region, they decide whether or not to enter into a possible food region and then search for a new food region so as to get high quality of nutrients. The bacterial foraging process consists of three main mechanisms: Chemotaxis, Swarming, Reproduction and Elimination-dispersal event. Chemotaxis is the process of simulating the movement of E.coli bacteria, which is carried in a flagella, through swimming and tumbling. The cell also repels a nearby cell in the sense that it consumes nearby nutrients and so it is not physically possible to have two cells at the same location. A bacterium in times of stress releases attractants to signal the bacteria to swarm together. After chemotaxis steps, a reproduction step is taken. Fitness value of bacteria is sorted in an ascending order.The least healthy bacteria eventually dies while each of the healthier bacteria (those yielding lower value of the objective function) asexually splits into two bacteria, which are then placed in the same location. This keeps the swarm size constant. Elimination event may occur due to sudden changes like a significant local rise of temperature or a part of them may move to other regions in the environment that will effect the behavior of bacteria heavily. The elimination and dispersal event destroys the performance of chemotaxis event but dispersal may place bacteria near good sources of food [174]. The main advantages of BFO are: (i) It is easy to implement, few parameters to adjust, computationally efficient, etc. (ii) The BFO algorithm is more adaptive and can be easily applied to real world optimization problems and is efficient for global search. (iii) It also avoids the chance of premature convergence. (iv) BFO is robust and flexible to implement. (v) Its performance is high with respect to speed of convergence, quality of solution and rate of success.

Table 2.3 describes the comparison between the above discussed Grid scheduling heuristic approaches w.r.t features. The comparison of the above described approaches depicts that BFO is better in comparison to other heuristic approaches as BFO is the only technique which is able to attain optimal scheduling decision by satisfying QoS services and can thus be applied to a Grid environment.

Table 2.3: Comparison of Different Heuristic Approaches

Heuristics/ Features	**Parameters**	**Convergence**	**Premature Convergence**	**Services**	**Local/Global Search**	**Optimization Problems**
Tabu Search	Less parameters	Guaranteed convergence	Prevent premature convergence	conceptual, Simpler, easy to implement, no special memory requirement	Low global search	Combinatorial optimization
Hill climbing	Less functions	No guaranteed	Prevent premature convergence	Simpler and straight forward	Local search	Simple Optimization Problem
Simulated annealing	Less Functions	Converge In asymptotic time	Premature convergence	Easy to code, robust heuristic	Local search	Combinatorial optimization Problems
Genetic Algorithm	More functions	No guaranteed	premature Convergence	No need analytical knowledge, easy to run and implement	Global search capability	Multi objective optimization
Memetic Algorithm	More functions	Guaranteed convergence	Less chance of premature Convergence	Flexible	Global search	Complex objective functions, non-linear multi objective Combinatorial optimization Problems
Ant Colony Optimization	Less functions	Guaranteed convergence	avoid the premature Convergence	Versatile, robust	Global search	Static and dynamic Combinatorial optimization Problems
Particle Swarm Optimization	No function like genetic algorithm	Slow convergence rate	Less chance of premature Convergence	Robust	Global search	Stochastic optimization
Bacterial Foraging Optimization	No function like genetic algorithm	Better convergence rate	Avoid premature Convergence	Flexible & Robust	Global search	Real-world optimization

2.4.4 Meta-heuristics

Meta-heuristics support in decision-making with robust tools that provide high-quality solutions to important applications in business, engineering, economics and science in reasonable time horizons [175]. Some of the advantages of meta-heuristics are: (i) It is an iterative master process that guides and modifies the operations of subordinate heuristics in order to produce high quality solutions. (ii) Meta-heuristics are very flexible to solve real problems. (iii) They are often used for global optimizers. (iv) It often robust to problem size, problem instance and random variables. Meta-heuristics have certain disadvantages which are as follows: (i) They perform well on a particular real-world problem but may not work on all problems. (ii) They may produce very poor solutions for other problems or even for other instances of the same problem. It requires extensive knowledge in both problem domain and appropriate heuristic techniques. (iv) They are quite expensive to implement. Meta-heuristics are not suitable in those situations when problems data and business requirements change frequently over time. (v) Optimality may not be guaranteed in meta-heuristics. (vi) There is a lack of theoretic basis and it requires multiple search parameters. (vii) Meta-heuristic algorithms like tabu search, ant colony etc have different searches but sometimes different searches may yield different solutions to the same problem.

2.4.5 Hybrid-heuristics

Hybrid strategies have been constructed to exploit the meta-heuristic techniques. To get a better result of the genetic algorithm, it has been hybridized with local search methods as tabu search, simulated annealing, etc. The advantage of parallel hybrids implemented on shared-memory parallel architectures is their simplicity [176]. The main advantage of hybrid-heuristic is that it has a better convergence. Hybrid-heuristics are more efficient in comparison to genetic algorithms. Some of the disadvantages are that hybrid-heuristics are not easy to implement and are time consuming.

2.4.6 Hyper-heuristics

The term Hyper-heuristics describe heuristics to choose heuristics in the context of combinatorial optimization. A hyper-heuristic can be seen as a high level methodology which when given a particular problem instance or class of instances and a number of low-level heuristics, automatically produces an adequate combination of the provided components to effectively solve the given problems [177]. This approach has several advantages: (i) Hyper-heuristics operate in a space of heuristics choosing and applying one low-level heuristic from a given set at each decision point. (ii) Hyper-heuristics do not require knowledge of each low-level heuristic, their working or the contents of the objective function of the problems. (iii) Hyper-heuristics are robustness and reapplicability heuristics. Some of the disadvantages of hyper-heuristics have been identified by Chakhlevitch K. et al. [178]. Some hyper-heuristic techniques make use of additional problem specific knowledge. Such knowledge can be used to describe the current state of the problem in order to select a suitable low-level heuristic in hyper-heuristics employing learning classifier systems. In indirect GAs, a portion of problem-specific information is often injected into the chromosome. For many hyper-heuristics, a significant amount of parameter tuning is required in order to find good parameter settings for a given problem. A large number of problem instances may be required for training and testing of the method in order to accumulate enough knowledge to make the right choice of low-level heuristics. However, in case of any real-world problem, the data may not be easily available and randomly generated instances may not adequately represent the real distribution. Many hyper-heuristic methods are only tested on a relatively simple benchmark problems for which the best solutions (often optimal) as well as an effective low-level heuristics are known in advance. There is no evidence that such hyper-heuristics would be effective in more complex real-world situations.

After the analysis of heuristic approaches, it has been analyzed that local heuristics may not be sufficient to find an optimal solution to combinatorial optimization problems. So, meta-heuristic techniques are used to solve such type of problems efficiently but it also requires extensive knowledge in both problem domain and appropriate heuristic techniques. Moreover, meta-heuristics are quite expensive to implement. As compared to meta-heuristic, hyper-heuristic can be seen as a high-level methodology [177]. The main difference between meta-heuristic and hyper-heuristic is that the former operates directly on the problem search space to find an optimal solution where as the latter finds the heuristic/problem solver to solve the problems. After the analysis of the heuristic approaches, it has been analyzed that hyper-heuristic is better in comparison to other heuristic approaches like meta-heuristics and hybrid-heuristics. Hyper-heuristic approach is more pliable to Grid environment where both the resources and jobs are highly diverse and dynamic in nature.

This section discussed about the Grid scheduling heuristic approaches and the next section discusses about QoS in Grids.

2.5 QoS in Grids

QoS requirements can be of various kinds due to the heterogeneous nature of the Grid resources. In Grid environment, QoS issues not only include network QoS and device QoS issues, but also include resource QoS issues [179]. With the emergence of distributed multimedia applications however, QoS has become a major issue in distributed systems research [180]. In distributed systems, QoS is applicable in the following areas described by Andrew Campbell et.al [180] (i) message passing services, which allows programmer to explicitly send a message between two or more processes in a distributed system (ii) remote invocation, which allows operations in a server process to be invoked by a client process and (iii) stream services, which are connections that support the transmission of continuous media flows [180]. QoS support is of paramount importance, given the inherent shared nature of Grid services and the limited capabilities of hardware and software resources available to satisfy client requests [181]. Sun H. X et al identified QoS policies based on resource sharing in grid computing in [181]. Along with the maturity of systems like Websphere, Gnutella, Skpe, Seti@home, Condor, PPlive and Globus, provision of QoS for these newly emerged computing platforms has also become one of the most important criteria [181]. In a shared network environment like Grid, there are several new issues related to QoS support that do not arise in a single computer system. He Sun. X et al has given resource availability, parallel processing and no-centralized control as the major issues. These issues of QoS make Grid computing extremely challenging. Providing non-trivial QoS is one of the primary goals of the Grid approach, as without such support, distributed resources becomes unusable [60]. In OGSA, the QoS characteristics of physical resource cannot represent that of logical resource. To convert the user's QoS request to particular QoS parameters that are in Grid becomes a problem which needs to be solved urgently [179].

2.5.1 QoS parameters

QoS can be categorized into various parameters. These parameters can be used for evaluating the Grid services. According to QoS parameters, user can easily evaluate the kind of services they are receiving.

QoS Parameter Categorization

QoS parameters categorization have been done by Kyriazis et al. in [182]. On logical basis, QoS parameters can be categorized as follows:

- User-defined Parameters: User defined parameters which relate to requirements / constraints that the user who initiates the workflow would like to pose, such as cost restrictions (e.g. maximum overall cost).

- Application Parameters: Application parameters which relate to the offered QoS from the application perspective. For example, the application configuration could play a significant role to the availability of the task to be executed.

- Resource Parameters: Resource parameters which relate to all types of resources, including computational, storage and network resources.

For Grids, QoS parameters are mainly related with resources like cost of the resources and resource utilization. Resources are of two types as defined by Xiaozhi Wang et.al in [179], soft resource and hard resources. Soft resources include data, information and software. Hard Resource includes devices like CPU, monitor, etc.

2.5.2 SLA in Grids

Managing QoS in distributed environment like Grid is a challenging task because SLA must be established and enforced both globally and locally at the components involved in the execution of tasks [6]. SLA is one of the most important elements of QoS in Grid computing. SLA defines a mutually agreed upon set of user's expectations and provider's obligations. SLA encodes QoS parameters such as resource availability, response time, advance reservation and completion deadlines [183]. The SLA is typically specified through a template containing both quantitative and qualitative information [184]. Due to the heterogeneous nature of the Grid environment, user must be given some form of commitment and assurances on the top of the allocated resources such as performance, security, availability, latency, throughput as well as in terms of responsibilities for dealing with invalid conditions fail over policies or backups and more. Those terms need to be agreed upon before use and manifested in the form of an SLA [185]. SLA allows clients to understand what to expect from the resource without knowledge of resource owner's policies [186]. SLA is a great tool to enforce different QoS policies in a Grid environment [181]. A detailed description of SLA components have been defined in [187]. Czajlowski K et al. defined three types of SLA's in [186]. Figure 2.3 describes the three types of SLAs as:

i. Task Service Level Agreements (TSLAs): In Task SLA, one negotiates for the performance of an activity or task.

ii. Resource Service Level Agreements (RSLAs): In Resource SLA, one negotiates for the right to consume a resource.

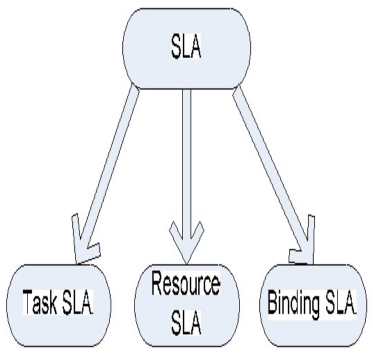

Figure 2.3: Types of SLA

iii. Binding Service Level Agreements (BSLAs): In Binding SLA, one negotiates for the application of a resource to a task.

2.6 Problem Formulation

Any distributed computing environment including Grid, needs to satisfy scalability, management and performance demands. Among what clearly distinguishes Grid from other distributed computing environments is its resource management requirements, which cannot be addressed by existing resource management mechanisms for distributed environments. It is also apparent from the literature survey that there are various mechanisms for Grid scheduling. The main features of the schedulers include advance reservation of the resources and job submission. In Grid environment, the resources are heterogeneous and can be geographically located all over the world. These resources i.e computing, storage, networking, scientific instruments can be accessed by using different policies. For the resource management technique to be efficient and in an understandable form, there is a need to associate resource provisioning with resource management i.e. there is a need to attach QoS parameters with resource provisioning and an efficient resource scheduling algorithm.

Based on the findings of the literature available and the trends in Grid computing, following problem has been formulated for this research work: Unless resource provisioning is considered a fundamental capability, predictable QoS cannot be delivered to Grid consumers. So it is required to design a resource provisioning policy based on QoS parameters for Grid environment. There is a need to design an efficient algorithm for scheduling the resources in a Grid environment, based on resource provisioning. This thesis attempts to propose and implement Resource

Provisioning and Scheduling Framework which provisions and schedules the resources in an effective way.

The main objectives of the proposed work are:

1. To analyze the Grid Resource Provisioning and Scheduling techniques.

2. To design QoS parameter(s) based Resource Provisioning Policy (RPP) for a Grid Environment.

3. To design an efficient resource scheduling algorithm based on above designed Resource Provisioning Policy.

4. To validate the proposed algorithm in a Grid Environment.

This chapter focused on exploring the existing resource provisioning and scheduling techniques in Grid environment. It analyzed existing resource provisioning strategies, scheduling in Grid middleware, Grid schedulers and heuristic approaches. The next chapter proposes a resource provisioning and scheduling framework to address the issues identified in problem formulation and to accomplish the objectives of this research work.