CHAPTER 2

LITERATURE SURVEY

A grid system is created using a lot of heterogeneous and homogeneous resources for dealing with the large-scale scientific problems. However, many issues are faced while using a grid system. To be able to take full benefit of a grid system, resource management and grid scheduling are the most important grid services, where issues of the job allocation, load balancing and fault tolerance represent a universal challenge for most grids. In the literature, many grid scheduling techniques, fault tolerance and load balancing methods have been proposed. Most of them can be applied to a computational grid with suitable modifications.

2.1 GRID SCHEDULING

Grid Scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains.

A job is defined as a program that needs a resource. The term resource is a hardware or software component that can be scheduled. The grid scheduler does not have control over the full set of jobs submitted to it (Jennifer Schopf 2004). Scheduling in its different forms is computationally hard; it has been shown that the problem of finding the optimum scheduling in heterogeneous systems is NP-hard (Jennifer Schopf 2004).
2.1.1 Grid Scheduling Characteristics

i. **Dynamic Structure of Computational Grid:** Unlike the traditional distributed systems, the resources in a grid can join or leave the Grid in an unpredictable way.

ii. **Resource Heterogeneity:** Computing resources may be very disparate in their computing power, ranging from laptops, desktops, clusters, supercomputers and even small computational devices.

iii. **Job Heterogeneity:** Jobs arriving at a grid are diverse and heterogeneous in terms of their computational needs.

iv. **Network Heterogeneity:** Grid resources are connected through the Internet using different interconnection networks.

v. **Local Schedulers:** Grids are constructed by the contribution of the computational resources by institutions, universities, enterprises and individuals. Most of these resources may eventually be running local applications and local schedulers.

vi. **Local Resource Policies:** It is because of the different ownership of the resources that one cannot assume full control over the grid resources.

vii. **Job-Resource Requirements:** Grid schedulers cannot assume full availability and compatibility of resources while scheduling jobs.

viii. **Large Grids:** Efficient resource management and the use of different types of schedulers are required to achieve the scalability for large-scale grids and a large number of jobs.
ix. **Security:** Can be seen as a two-fold objective: on the one hand, a job has security requirements and on the other hand, the resources have their own security requirements.

### 2.1.2 Classification of Schedulers (Vivekanandan and Ramya Chitra 2011)

Grid Scheduler is a software component that is in charge of computing a mapping of jobs to resources under multiple criteria and grid environment configurations. Different levels of a grid scheduler are:

i. **Super-Scheduler:** It corresponds to a centralized scheduling approach in which the local schedulers are used to reserve and allocate the resources in the grid. The super-scheduler is in charge of managing the advance reservation, negotiation and service level agreement.

ii. **Meta-Scheduler:** Also known as a meta-broker arises when a single job is allocated to more than one remote resource. Meta-schedulers coordinate the local schedulers of the remote resources to compute an overall schedule.

iii. **Local/Cluster Scheduler:** Assigns jobs to resources in the same local area network. The scheduler manages the local resources and the local job queuing system.

iv. **Enterprise Scheduler:** Used in large enterprises having computational resources distributed in many enterprise departments. The enterprise scheduler uses different local schedulers belonging to the same enterprise.
v. **High-Throughput Schedulers:** Its objective is to maximize the average number of jobs processed per unit of time.

vi. **Resource-Oriented Schedulers:** Its objective is to maximize the resource utilization.

vii. **Application-Oriented Schedulers:** Are concerned with the scheduling applications in order to meet a user's performance criteria.

### 2.1.3 Classification of Scheduling Techniques (Hui Yu et al 2005)

i. **Immediate Mode Scheduling:** In immediate mode scheduling, jobs are scheduled as soon as they enter the system.

ii. **Batch Mode Scheduling:** In batch mode scheduling, jobs are grouped into batches which are allocated to the resources by the scheduler. The results of the processing are usually obtained at a later time.

iii. **Non-Preemptive Scheduling:** In the non-preemptive scheduling, the job should entirely be completed by the designated resource.

iv. **Preemptive Scheduling:** The current execution of the job can be interrupted and the job is migrated to another resource. Preemption can be useful if the job priority is to be considered as one of the constraints.

v. **Cooperative Scheduling:** In cooperative scheduling, a feasible schedule is computed through the cooperation of the procedures, rules and grid users.
2.1.4 Grid Scheduling Phases (Jennifer Schopf 2004)

There are three main phases involved in grid scheduling namely: Resource discovery, information gathering and job execution.

Figure 2.1 Three Phase Plan for Grid Scheduling (Jennifer Schopf, User as a Grid scheduler)

2.1.4.1 Phase 1: Resource Discovery

The first stage in any scheduling interaction involves determining which resources are available to a given user. The resource discovery phase as shown in Figure 2.1 has three steps: Authorization filtering, job requirement definition and filtering to meet the minimal job requirements.

i. **Authorization Filtering:** The first step of the resource discovery for the grid scheduling is to determine the set of resources based on specific constraints. At the end of this step the user will have a list of machines or resources to which he or she has access.
ii. **Application Requirement Definition:** To proceed with the resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources.

iii. **Minimum Requirement Filtering:** The third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. At the end of this step, the user acting as a grid scheduler will have a reduced set of resources to investigate in more details.

### 2.1.4.2 Phase 2: System Selection

i. **Dynamic Information Gathering:** For making the best possible job resource match, detailed dynamic resource information is collected which varies with respect to the application being scheduled and the resources being examined. The dynamic information gathering step has two components: what information is available and how the user can get access to it.

ii. **System Selection:** Based on the information gathered, the systems matching the job requirements are selected.

### 2.1.4.3 Phase 3: Job Execution

i. **Job Submission:** Once the resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated as running a series of scripts and may or may not include setup or staging.
ii. **Preparation of Tasks:** The preparation stage may involve the setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application.

iii. **Monitoring Progress:** Depending on the application and its running time, the users may monitor the progress of their applications and possibly change their mind about where or how it is being executed.

iv. **Job Completion:** When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines include an e-mail notification parameter.

v. **Clean-up Tasks:** After a job is being run, the user may need to retrieve files from that resource in order to perform data analysis on the results and remove the temporary settings.

2.1.5 **Types of Grid Scheduling (Fatos Xhafa and Ajith Abraham 2010)**

i. **Independent Scheduling:** The potential of a massive capacity of a parallel computation is one of the most attractive characteristics of the computational grids. The common characteristic is the partitioning of jobs into sub-jobs which can be scheduled independently.

ii. **Grid Workflows:** Solving complex problems in grids requires the combination and orchestration of several processes. This arises owing to the control and data dependencies in the solution flow. This class of applications is known as grid workflows.

iii. **Centralized, Hierarchical and Decentralized Scheduling:** Both the centralized and the decentralized scheduling differ in the control of the resources and the knowledge of the overall
grid. In centralized scheduling, the scheduler has knowledge of the system by the monitoring of the resource state. The centralized scheduling suffers from a limited scalability and a single point of failure and is thus not appropriate for large scale grids. Organizing the grid schedulers in hierarchically structure allows the co-ordination of different schedulers at each level. In this case, schedulers at the lowest level in the hierarchy have knowledge of the resources. This scheduler type still suffers from lack of scalability, yet it scales better and is more fault tolerant than the centralized schedulers. In the decentralized or distributed scheduling there is no central entity which controls the resources. The scheduling requests, either by the local users or other grid schedulers, are sent to the local schedulers which manage and maintain the state of the job queue. This type of scheduling is more realistic for real large-scale grid systems.

iv. **Static versus Dynamic Scheduling:** There are essentially two main aspects that determine the dynamics of the grid scheduling, namely; (a) the dynamics of jobs and (b) the dynamics of resources. In static scheduling, there is no job failure and resources are assumed to be available all the time. Although this is unrealistic for most grids, it is used in the batch mode scheduling.

v. **Adaptive Scheduling:** The changeability of the grid environment requires the adaptive scheduling techniques to consider both the current status of the resources and predictions for the future status with a view to detect and avoid performance deterioration. Rescheduling can also be
seen as a form of adaptive scheduling in which running jobs are migrated to the more suitable resources.

2.1.6 Existing Works

The FPLTF (Fastest Processor to Largest Job First) (Saha et al 1995) algorithm schedules jobs to resources according to the load of jobs in the grid system. The grid scheduler sorts out the jobs and resources by their load and CPU speed then assigns the largest job to the fastest available resource. If there are many jobs with heavy load, its performance may be very bad. Min-min (Maheswaran et. al 1999) sets the jobs which can be completed at the earliest with the highest priority. The main idea of Min-min is that it assigns jobs to the resources which can execute jobs faster. Max-Min (Maheswaran et al 1999) sets the jobs which have the shortest completion time with the highest priority. The main idea of Max-Min is that it overlaps the jobs with long running time with the jobs with short running time. For instance, if there is only one long job, Min-Min will execute short jobs in parallel and then execute long jobs. Max-Min executes short jobs and long jobs in parallel.

The FCFS (First Come First Serve) algorithm is a simple job scheduling algorithm. A job which makes the first requirement is executed first. The main problem of FCFS is its convoy effect (Abraham Silberschatz et al 2005). Convoy effect is short jobs waiting behind long jobs in a queue. If all jobs are waiting for a big job to finish, the convoy effect occurs. The convoy effect may lead to the longer average waiting time and lower resource utilization. The Most Fit Task First (MFTF) algorithm proposed by (Sheng-De Wang et al 2005) attempts to discover the fitness between jobs and resources for the user. However, MFTF does not consider the resource utilization and the estimation function is an ideal method. Hence the incorrect scheduling may occur in the real environment.
Braun et al (2001) proposed Minimum Execution Time (MET) and Minimum Completion Time (MCT) scheduling algorithms. But both MET and MCT lead to poor makespan. Xhafa et al (2007) examined a subset of these methods on an offline system.

Sugavanam et al (2007) and Shestak et al (2008) proposed the resource allocation heuristics, but the approaches did not guarantee that jobs would be assigned to machines with their fastest execution times. Mehta et al (2007) proposed the dynamic resource allocation heuristics to manage the trade-off between makespan and robustness. One major disadvantage of this method is that a computing resource with low processing capability may never be selected and will always remain idle or less loaded. Hui et al (2005), Ruay-Shiung Chang et al (2009) proposed ant colony optimization algorithm for job scheduling. However, they considered the pheromone value to be the sum of different units which affected the accuracy in the result of the scheduling algorithm. Yun-Han Lee et al (2011) proposed hierarchical framework and job scheduling algorithm to balance the system load and minimize the makespan of jobs. However, Lee neglected the job length and assumed that the cluster is static.

2.2 LOAD BALANCING

Grids functionally unite worldwide scattered computers for creating a universal source of computing power and information. A key trait of grids is that the resources are shared among numerous applications and the quantity of resources available to any given application highly fluctuates over time (Watts and Taylor 1998).

Therefore, load balancing is a technique to augment resources, utilizing concurrency, exploiting throughput improvisation and cutting
response time through an appropriate distribution of application. Using multiple components with load balancing instead of a single component may increase the reliability through redundancy (Elsasser et al 2002).

The importance of the load balancing is as follows:

i. An unexpected peak load can be routed to the relatively unoccupied grid system.

ii. If the grid is fully utilized, then the lowest priority job currently being performed can be either temporarily suspended or cancelled and performed later so as to make room for the higher priority jobs.

2.2.1 Load Balancing in Grid Environment and DCE

Often, grid computing is regarded as a successor of the distributed and parallel computing. Nevertheless the grid computing and distributed computing environment (DCE) are elementarily different. A DCE is predictable: The availability of resources is based on the fact that the reservation and processing speeds are static and known in advance. A grid environment, nonetheless, is greatly unpredictable as the resources have dissimilar and unknown processing speeds and they can be added or removed at any time. As a result, the dynamic nature of the grid makes load balancing a challenge.

2.2.2 Load Balancing Characteristics

In designing load balancing algorithms, some essential characteristics should be kept in mind. They are as follows (Simone et al 2011):
i. **Optimum Resource Utilization:** A load balancing algorithm should optimize the utilization of the resources by optimizing the time or cost related to these resources. Since the grid environment provides a dynamic search space, this optimality is inexorably a partial optimality of the performance.

ii. **Fairness:** A load balancing algorithm is said to be fair, meaning that the difference between the heaviest loaded computing resource and the lightest loaded computing resource in the network is minimized, keeping in mind that the search space is dynamic.

iii. **Flexibility:** As the topology of the network or grid goes on changing, the algorithm should be flexible enough to adhere to the changes.

iv. **Robustness:** Robustness refers to the fact that in case of failures, the algorithm should have a way to deal with the failure coping with the situation.

v. **Distribution:** Distribution for managing resources and running the load balancing algorithm has the benefit of leaving out a single point of failure.

vi. **Simplicity:** The sizes of the software units are important as they take up bandwidth when they want to transfer themselves between the resources. Since these units are executed on the grid resources, there is a preference to keep the necessary computations as simple as possible.
2.2.3 Load Balancing Algorithms: A Classification (Osman et al and Ammar 2004)

Initiation strategy is based on the load balancing actions. Periodic initiation is a timer-based initiation in which the load information is exchanged over fixed time intervals. Event-driven initiation is generally a load dependent strategy based on load surveillance.

Event driven initiation can further be classified into sender-initiated policy and receiver-initiated policy. In the sender-initiated policy, the congested nodes move their load to the lightly loaded nodes. In the receiver-initiated policy, the lightly loaded nodes reduce the workload of the heavily loaded nodes.

Sender-initiated policy performs better than the receiver-initiated policy at low to moderate system loads as shown in Figure 2.3. This is because, the probability of finding a lightly loaded computing resource is higher than that of finding a heavily loaded computing resource at low and moderate loads. Similarly at high system loads, the receiver initiated policy performs better since it is much easier to find a heavily loaded computing resource. The adaptive policies behave like the sender-initiated policy at low to moderate loads and the receiver-initiated policy at high loads.
Figure 2.2 Classification of Load Balancing Strategies
Figure 2.3 Sender Initiated Approach vs. Receiver Initiated Approach

The load balancer location strategy stipulates the location where the algorithm itself is executed. It is central as shown in Figure 2.4, if it is executed on a single processor, determining the essential load transfers and informing the concerned processors. If every processor takes part in the load balancing operation, the algorithm is categorized as decentralized.

Figure 2.4 Centralized Load Balancing
Information exchange strategy stipulates the information and load flow through the system. The information employed by the dynamic load-balancing algorithm for decision-making can be the local information on the processor or collected from the vicinity. The global information exchange strategies tend to provide the extra precise decisions, and the local information exchange strategies may capitulate to less communication costs. The communication policy stipulates the connection topology of the processors in the system that decides the neighbourhood of every processor. The communication policy also indicates the job or load exchange among different processors. The local strategies describe a group of processors, and allow transfers to take place just between processors within the similar group, while in global strategies, job or load transfers may take place between any two processors.

Load selection strategy stipulates the processors involved in the load exchange (processor matching) specifying the suitable load items (load matching) to be exchanged.
2.2.4 Load Balancing Policies

The four policies that govern the action of a load balancing algorithm when a load imbalance is detected deal with information, transfer, location and selection. The information policy is responsible for keeping up-to-date load information about each resource in the system.

The transfer policy deals with the dynamic aspects of a system. It uses the resources’ load information to decide when a resource becomes eligible to act as a sender (transfer a job to another resource) or as a receiver (retrieve a job from another resource).

Location policy selects the partner resources for a job transfer transaction. If the resource is an eligible sender, the location policy seeks out a receiver resource to receive the jobs. If the resource is an eligible receiver, the location policy looks for an eligible sender resource. Once a resource becomes an eligible sender or receiver, a selection policy is used to pick up the queued jobs to be transferred. A stable symmetrically initiated adaptive algorithm uses the information gathered during polling to classify the resources as overloaded, under-loaded or OK (resources having manageable load). These actions impose a small and constant overhead, irrespective of the number of resources in the system.

Selection policy uses several criteria to evaluate the queued jobs. Its goal is to select a job that reduces the local load, incurs as little cost as possible in the transfer and has good affinity to the resource to which it is transferred. A load balancing algorithm in which a computing resource exchanges information and transfers jobs to its physical and/or logical neighbours is called “neighbour-based” load balancing method. The load balancing algorithms in which the computing nodes are partitioned into
clusters based on the network transfer delay are called “cluster-based” load balancing methods.

2.2.5 Load Balancing and Scheduling

Load balancing based scheduling can be classified into queues and rescheduling.

2.2.5.1 Queues

There are two types of queues in grid environment. The internal queue of a computing resource consists of the ready jobs to be executed by the particular computing resource only. The external queues of a computing resource consists of the jobs initially submitted to this computing resource by a user but are yet to be mapped and scheduled for execution.

2.2.5.2 Rescheduling

Over time, other applications may introduce load on the system or application requirements may change. It is to sustain a good performance for longer running applications that the schedule needs to be modified during the application execution. This process called rescheduling can include changing the machines on which the application is executing (migration) or changing the mapping of data and/or processes to those machines (dynamic load balancing).

2.2.6 Existing Works

Lin and Keller (1987) presented a demand driven gradient model for load balancing. In this approach a job continues to be transferred until it reaches an under-loaded computing resource or a computing resource for which no neighbouring computing nodes report the shortest distance to the
nearest lightly loaded computing resource. Cybenko (1989) put forward an adaptive contracting within the neighbourhood (ACWN) method which applied a saturation control technique on the number of hops that the job had to travel before reaching its destination. ACWN performed consistently better than the gradient model but did not take into account a computing resource’s own load before making a job migration decision.

Cybenko (1989) first introduced diffusion as a method for load balancing. Willebeek-LeMair and Reeves (1993) also explored diffusion and proved that it is superior to other load balancing strategies in terms of its performance, robustness and scalability assuming that no new workload is completed during the execution of the algorithm.

Cheng Zhong Xu and Francis (1994) proposed a neighbour-based iterative dynamic load balancing algorithm for multi computers in which processes migrate one step at a time. Migration at each step is performed according to a local decision made by the intermediate processor. Xu and Lau (1994) derived optimal parameters that maximize the convergence rate on mesh, torus and N-D hypercube. Qian and Yang (1991) showed that without the quiescent assumption in (Cybenko 1989), it is possible to prove that the variance of the unbalanced workload is bounded.

Song (1994) proposed a partially asynchronous and iterative algorithm for distributed load balancing. A major disadvantage of the diffusion approach is that it requires much iteration to achieve load balancing. Watts Taylor et al (1998) overcame this by using a fully implicit diffusion scheme with adaptive time steps. Another improved diffusion algorithm was derived based on Chebyshev polynomials which showed significantly faster convergence than baseline diffusion method, but at the additional cost of calculating two Eigen values (Hu et al 1999).
Nishimura et al (2001), Elsasser et al (2002) proposed the load balancing schemes for computational grid environments. However, both of them neglected the overhead involved in collecting the state information while balancing the load. Grosu et al (2005) proposed a non-cooperative load balancing game for the distributed systems, but did not consider the communication delays in a grid environment. Kai Lu et al (2006 and 2007) proposed a decentralized load balancing scheme for computational grid environment namely DA which takes into account the concerns of scalability, site heterogeneity and significant communication overheads. In order to minimize the overhead of information collection, the state information exchange is done by the mutual information feedback (MIF). However, Kai Lu did not consider the execution scheme for data distribution. Riky Subrata et al (2008) proposed the artificial life techniques for balancing the load of computational grids. However, this approach incurs an extra processing requirement at each scheduling resource.

Ruchir et al (2007) presented a modified ELISA algorithm that considers the job migration cost which is primarily influenced by the available bandwidth between the sender and the receiver computing nodes while taking decision for load balancing. Modified ELISA algorithm did not achieve a better utilization of the powerful processing computing nodes in a heterogeneous system. Riky Subrata et al (2008) proposed a game-theoretic approach to load balancing. The load balancing problem is modelled as a non-corporative game, where in the objective is to reach the Nash equilibrium. It is a strategy profile where in no scheduler can decrease its average job completion time by unilaterally changing its strategy. There exists a unique Nash equilibrium because the expected response time functions are continuous, convex and increasing. Game-theoretic approach employs periodic status information exchange strategy and the schedulers update their information in a sequential manner.
Suri and Singh (2010), Chatrapati et al (2010) proposed load balancing schemes for grid environment but neither the algorithms follow the changes in the system status nor set fixed threshold for controlling the load. Malarvizhi and Rhymend Uthariaraj (2011) proposed a decentralized load balancing algorithm for computational grid in which the load updates between resources are done periodically leading to a high messaging overhead.

Malarvizhi and Rhymend Uthariaraj (2011) demonstrated a sender-initiated decentralized dynamic load balancing scheme for the multi-cluster computational grid environment (SI-DDLB) which is an extended study of the approach proposed in (Kai Lu et al 2006 and 2007). However, SI-DDLB neither models the impact of the accuracy of the job execution time estimation nor considers the execution scheme for data distribution.

2.3 FAULT TOLERANCE

Resource failures (processors/links) may frequently occur in grid systems causing an adverse effect on the applications. Consequently, there is an increasing need for developing techniques to achieve fault tolerance (Song 1994). In multiprocessor systems, the fault tolerance can be provided by scheduling replicas of jobs on different processors. There are two main approaches to replication as described below.

2.3.1 Active Replication

This technique is based on the space redundancy i.e. multiple copies of each job are mapped on different processors and run in parallel to tolerate a fixed number of failures. With such a technique, no fault detection mechanism is required.
2.3.2 Passive Replication

The main idea of this technique is that a backup copy of a job is activated only if a fault occurs while executing its primary copy. Passive replication scheme does not require fault diagnosis and it guarantees to recover all failed jobs. In such a scheme, only two copies of the job are scheduled on different processors. Two techniques can be applied while scheduling primary and backup copies of each job. (1) Backup overloading consists of scheduling backups for multiple primary jobs during the same time slot in order to efficiently utilize the available processor time; and (2) De-allocation of the resources reserved for the backup jobs when the corresponding primaries are completed successfully.

2.3.3 Existing Works

Many fault tolerant schemes have been proposed for grid systems (Manimaran and Murthy 1988, Al-Omari et al 2001, Abawajy 2004). Ghosh et al (1997) introduced the backup overloading to reduce the replication cost of the independent jobs. However, he could not guarantee to find an optimal schedule for the backups of the independent jobs in terms of the replication cost. Luo et al (2007), Quin Zheng et al (2009) proposed a dynamic and reliability-driven real-time fault-tolerant scheduling algorithm. Xiaomin et al (2011) proposed QoS-aware fault-tolerant scheduling algorithm for heterogeneous clusters. However, they focused on a centralized scheduler model and did not take into consideration the network bandwidth while scheduling the real-time jobs.