Chapter 3

Design of Grid Scheduler

The scheduler component of the grid is responsible to prepare the job que for grid resources. The research in design of grid schedulers has given various topologies for grid design. We discuss architectures of the Grid schedulers and related issues. The decision making process as is based on GIS and Scheduling parameters, the discussed architectures carries benefits as well as flaws concerned to use of scheduler. The issues which influence the scheduling hierarchies are elaborated in section 3.2. A comparison of different available scheduling architectures is presented in section 3.3. OGSSA follows a two level scheduling hierarchy and the reasons behind the adoption of this approach are stated in section 3.4. The OGSSA scheduling architecture and scheduling API are presented in the section 3.5.

3.1 Introduction

The Grid concept was created to facilitate the use of available distributed resources effectively and efficiently. The first step needed before one can utilize the Grid for running jobs is to locate and use (the best) resources available to serve those jobs that is the process of resource scheduling. The observation is that centralized algorithms and environments are less effective than their decentralized counterparts when scheduling data intensive bulk jobs. This chapter will also evaluate that applying the concept of P2P systems to resource scheduling can lead to efficient resource utilization [88]. A meta-scheduler coordinates the communication between multiple heterogeneous local schedulers that typically manage clusters in a LAN environment. In addition to providing a common entry point, a meta-scheduler [15] also enables global access and coordination, whilst maintaining local control and ownership of resources through the local
Schedulers. The fundamental difference between a meta-scheduler and local schedulers is that a meta scheduler does not own the resources and has no autonomy in its decisions. Therefore, the meta-scheduler does not have total control over the resources. Furthermore, a meta-scheduler does not have control over the set of jobs already scheduled to a local scheduler (also referred to as the local resource management system). These local and meta-schedulers form a hierarchy and individual schedulers sit at different levels in the hierarchy as discussed by in [74]. Each local scheduler can cooperate and communicate with its siblings through a meta-scheduler, however, each meta-scheduler cannot communicate with other meta-schedulers of other sites or Grids. Communication is only possible between local schedulers and the meta-scheduler. Existing scheduling systems are often based on the client-server architecture with one or several meta-schedulers on top of independent local schedulers such as LSF, PBS [83] etc. Each local scheduler can collect information and can schedule the jobs within its own managed site. Typically, these local schedulers cannot schedule jobs to some other available site.

Peer-to-Peer (P2P) scheduling systems on the other hand can provide environments where each peer can communicate with all other peers to make global decisions at each site, can propagate their information to other peers, and can control their behavior through this information. In the P2P approach, a meta-scheduler and a local-scheduler make a hierarchy at each site where global decisions are managed by the meta-scheduler whereas local control and allocations are made by the local scheduler. The meta-scheduler on each site has access to global information and all meta-scheduler instances communicate with each other to share the cost and load information. Our intention is to incorporate a P2P approach so that schedulers do not take global decisions at a single central point, but rather many sites participate in the scheduling decisions through sharing the information in their cost matrices. Each site should have information on load, queue size etc., should monitor its processing nodes and then propagate this information to other peers. Local and certain global policies could be managed at the site level instead of a central hierarchical management. In contrast to this approach, centralized scheduler management can be problematic in several ways since load balancing, queue management, job allocation, policies etc. are central and are typically managed by a (single) central meta-scheduler and might not be fault tolerant. Note that by client server architecture, we do not mean here a tier system which uses various tiers, which are clients of each other, to scale up the client server behavior. Each tier is not scalable if treated in isolation. Schedulers may be subject to failure or may not perform efficient scheduling when they are exposed to millions of
jobs having different quality of service needs and different scheduling requirements. They may not be able to re-organize or export scheduled jobs which could result in large job queues and long execution delays. For example in High Energy Physics (HEP) analysis a user may submit a large number of jobs simultaneously (the so-called bulk job scheduling), and the scheduling requirements of bulk jobs may well be different to those of singly queued jobs. In bulk job submission by a single or multiple users at a particular site it might become impossible for a local scheduler to serve all the jobs without using some job export mechanism. In the absence of this mechanism, it is possible that some of the jobs might be lost by the scheduler due to timeouts before they get an execution slot, insufficient space in the queue to hold the jobs or the fact that the frequency of submission cannot be handled by the central scheduling site. What is required is a decentralized scheduling system which not only automatically exports jobs to its peers under potentially severe load conditions (such as with bulk jobs), but at the same time it manages its own scheduling policies, whilst queuing jobs and monitoring network conditions such as bandwidth, throughput and latency. The queuing mechanism that is needed at each scheduling peer should follow the same queue management scheme across all the sites in order to enforce uniform scheduling policies across the Grid sites. This will enable sites to interact and export jobs to other sites without any policy conflicts since all sites are following a similar scheduling approach. It should associate priorities to each job inside the queue, depending on the user profile and the job requirements with the scheduler servicing high priority jobs preferentially to optimize Grid service standards. In this chapter, we explain the functionality of the scheduling models and present its scheduling and queue management mechanism and demonstrate the advantages and drawbacks of such a system implementation.

3.2 Considerations for Scheduling Hierarchies

The following sections describe the considerations for selecting an appropriate meta scheduler hierarchy. The choice of a particular scheduling paradigm is dictated by certain criteria which are listed below. Whether a centralized, hierarchical or a distributed scheduling hierarchy is selected, each approach has drawbacks and certain advantages over the other approaches. We explain the issues associated with the selection of the scheduling paradigm and then provide details of various scheduling hierarchies in the next section on the basis of these criteria.
3.2.1 Performance

When selecting a job scheduler and associated resources for the execution of the job, the Grid throughout and the performance requirements of the application must be considered. In most cases, an increase in Grid throughput will lead to a better performance of the applications and vice versa. The service requester is interested in a quality of service (QOS) that includes an acceptable turnaround time [41]. While building a Grid and exposing one or more applications as a service over this Grid, the service provider would like to maximize the utilization of the resources and the throughput of the systems within the Grid to get a better return on the investment and to provide a better quality of service. Therefore a performance Grid is equally important for both the service requesters and the service providers. It becomes necessary for the scheduler to better decide the sites which can guarantee better performance. Hasher et al. [8] state that the order and hierarchy of the schedulers is critical for the selection of the better performing sites.

3.2.2 Reliability

Availability of resources is a prime decision criterion for selecting a suitable scheduling hierarchy and it may have an influence on the scheduling optimization. The hierarchy should support the cases where the particular site or a scheduler becomes unavailable or where a scheduler might decide to migrate or to checkpoint a job. Resource availability is not only concerned with the choice of the sites and schedulers but it also deals with the execution software and environmental requirements for the jobs and the programs. When deciding the scheduling hierarchies, reliability should be the prime consideration so that the hierarchies can deal with the host and network interruptions as well as being able to optimize the scheduling decisions. There are many approaches [13] that can be considered to make the schedulers and applications reliable, for example, using checkpoint-restart mechanisms, having persistent storage to hold queue states in the scheduler and providing robust systems management solutions to maximize the availability of the Grid.

3.2.3 Scalability

Modern day applications are not only compute and data intensive but also demand a high level of scalability. A system should be easily scaled to accommodate changes in the number of
users, resources and computing entities affected by it. Scalability can be measured in three different dimensions: load scalability, geographic scalability and administrative scalability. A Grid system should support load scalability by making it easy for the users to expand and contract its resource pool to accommodate heavier or lighter loads. A geographically scalable system is one that maintains its usefulness and usability, regardless of how far apart its users or resources are. In administrative scalability, no matter how many different organizations need to share a single distributed system, it should still be easy to use and manage. Some loss of performance may occur in a system that allows itself to scale in one or more of these dimensions. There is a limit up to which we can scale/add processors to the system, and above which the performance of the system degrades. This feature should be considered in the scheduling system and the scheduling hierarchies should sense the network, load and other characteristics to make the scheduling decisions as described in [25]. Therefore before designing or enabling a Grid application, one should anticipate the scalability of the system.

3.2.4 Scheduling Policies

A scheduling policy is a criterion which dictates the way the jobs are allocated to resources for execution. This is followed in order to allow certain users and processes to grant priority to others at certain points in time. The need for scheduling policies arises from the requirement to support different execution scenarios. For example, there is a scheduling policy that a job might not be started immediately and it might be interrupted or preempted during execution; for example it might be scheduled to run overnight. Another scheduling policy might be to allow only 50 percent resources for a particular V0 with the remaining resources being reserved for the local use. A job can be asked to run or to support a limited set of job categories or support a middleware which is installed on particular fire-walled hardware and this might be intended for scheduling and execution optimization [97]. Consequently scheduling policies can have a profound effect on the scheduling hierarchies in determining the QoS [58] and scheduling decisions for sending and retrieving jobs to and from Grid sites clearly depend on the scheduling policy for that site. The choice of the job policies and priorities is central to scheduling optimization; open execution of the jobs without any scheduling policy will deprive certain users from a sustained QoS and accordingly will alter the preferences in selecting a scheduler [48].
3.3 Scheduling Architectures

Almost every job scheduling hierarchy has some benefits as well as limitations and these hierarchies are widely recognized with minimal variations. When submitting jobs for execution and scheduling and enabling applications for a Grid environment, the scheduling system needs to be aware of how the load-balancing mechanism (whether manual, push, pull, or some hybrid combination) will affect the application, specifically its performance and turnaround time. In this section we list major scheduling hierarchies and compare the pros and cons of each approach. This section will help explain the reasons which led us to select the P2P scheduling model.

3.3.1 Master/Agent Architecture

This is a centralized architecture for job schedulers and is suitable for the local or clustered scheduling. The Job Scheduling software is installed on a Master node while on the production machines only a very small component is installed (Agent) that awaits commands from the Master, executes them, and returns the results back to the Master. Condor [110], Sun Grid Engine[53], PBS [32] and others are based on this scheduling architecture. Figure 3.1 shows Centralized Scheduler. In a centralized scheduling hierarchy all production machines are scheduled and managed by a central Master instance. Information on the state of all available systems is also collected by the Master node. The problem with this approach is that a centralized scheduling hierarchy cannot scale well with increasing size of the computational Grid. The central scheduler may prove to be a bottleneck in some situations. For instance if a network error cuts off the scheduler from its resources, system availability and performance will be affected. Another disadvantage is communication overhead in collection of status from all production machines to a single central point.

As an advantage, the scheduler is conceptually able to produce very efficient schedules, because the central instance has all necessary information on the available resources.

3.3.2 Push and Pull Model

When a job is submitted to a Grid scheduler, its workload can be distributed in a push model, pull model, or combined model. For example, the gLite workload management system implements both push and pull scheduling policies. A round-robin scheduler basically implements the push model of the scheduling. However, the push model does not consider the job queue
lengths as discussed in [57] and forms a hierarchical architecture as shown in Figure 5.2. In the pull model, synchronization and serialization of the job queue will be necessary to coordinate the pulling of jobs by multiple Grid resources [74]. Local and global job queues are also managed in the pull model. Fail over conditions need to be considered in both of these scheduling models. Therefore, the monitoring system should detect the non-operational Grid resources and no new work should be sent to failed resources in the push model. In addition, all the submitted jobs that did not complete their execution need to be taken care of in both the push and pull models. All the uncompleted jobs in the failed host need to be either redistributed or taken over by other operational hosts in the virtual organization.

3.3.3 A Peer-to-Peer (P2P)-based Cooperative Scheduling Architecture

In a P2P scheduling model, each machine is capable of helping with the scheduling process and can offload locally scheduled jobs to other cooperating machines. In decentralized systems, distributed schedulers interact with each other and commit jobs to remote systems and no central instance is responsible for the job scheduling. Therefore, information about the state of all systems is not collected at a single point. Thus, the communication bottleneck of centralized scheduling is prevented which makes the system more scalable [88]. Also, the failure of a single component will not affect the whole scheduling system as is shown in Figure 3.2. This provides better fault-tolerance and reliability than is available for centralized systems without fall-back or high-availability solutions. But the lack of a global scheduler, which knows about all jobs
and system information at every time instant, could lead to suboptimal schedules. Nevertheless, different scheduling policies on the local sites are still possible. Furthermore, site-autonomy for scheduling can be achieved easily since the local schedulers can be focused on the needs of the resource provider or the resource itself. [9]

3.4 Hierarchies of Schedulers

In this section the peer to peer scheduling mechanism is discussed and its queue management process is elaborated upon. The peer to peer process to communicate and share the information between the meta-schedulers is illustrated and how they manage the jobs at a local and global level is highlighted.

A user submits a job to a meta-scheduler (local to the user, typically at the same site) which in turn contacts a local scheduler. A particular meta-scheduler considers only its own managed sites to schedule the job and does not look around for other sites managed by other schedulers to distribute load and to get the best available resources. The jobs are scheduled centrally irrespective of the fact that this may lead to a poor QoS due to potentially long queues and scheduling delays. Hence, the architecture with non-communicating meta schedulers (see Figure 3.4) can lead to inefficient usage of Grid resources.

Furthermore, in this architecture the meta-scheduler schedules the job on its site, and it
cannot communicate with the sibling meta-schedulers and hence does not consider the underlying network and data transfer costs between the sites. This is one of the reasons that almost all Grid deployments have at most only a few meta-schedulers and that any two cannot communicate and inter operate with each other. Peer-to-Peer (P2P) systems, on the other hand, provide environments where each peer can communicate with all other peers to make the global decisions at a site level, can propagate its information to other peers, and can self-organize and control themselves using this information. This feature can make scheduling decisions and resource utilization more efficient [113]. Our intention is to incorporate a P2P approach for performing the scheduling process. Schedulers should not take global scheduling decisions at a single central point, rather all sites should participate in the scheduling decisions. Each scheduler at a site should monitor its resources and then propagate this information to other peers in the Grid. Each site should broadcast its load and queue size to other peers; local and global policies should be managed at the same level instead of a hierarchical management. As a result, this P2P behavior can become a foundation stone for a self-managing, self-organizing and self healing resource management and scheduling system. For DIANA a P2P decentralized and self-organizing scheduling system is required which cannot only automatically export jobs to its peers under severe load conditions, but it can also manage its own scheduling policies, hierarchy, queue and network conditions. Each peer should follow some queue management scheme which can associate priorities with each job inside the queue and can ensure Grid service standards.
3.5 OGSSA Scheduler Design

This section elaborates the design of scheduler for OGSSA. The design is inspired from hierarchical scheduler. The global level and local level scheduling is performed with flexibility to use selected algorithm to prepare schedule. The Global level scheduler deals with application level scheduling. The number of applications submitted are given for scheduling to Global level algorithm. Further Local level scheduler is collection of Computing Elements (CE) and Storage Elements (SE). This collection is visualized as cluster of resources. Figure shows the hierarchical scheduler with

Global and Local levels. The Local level scheduler decomposes application received from Global level into set of jobs. These jobs are input to the Local level algorithm. For instance Global Level algorithm is RR and Local Level algorithm is FCFS then this is treated as on combination of algorithms.

3.5.1 Usecase Representation of Scheduler

3.5.2 Modules and Subsystems

The modules comprising the scheduler are described below: The scheduler runs as server and is responsible for hosts of tasks. They are as follows:
1. Obtaining CPU utilization from the clients

The very first action to take place is the reading of the CPU utilization of the various s
help of the SNMP module installed with them. This CPU utilization is the average of
the utilization values of the computers over the last 60 seconds. Once the Grid Scheduler
has these values, it will know the status of its resources and can accordingly proceed to
schedule jobs.

2. Enabling user to submit applications.

The Grid Scheduler provides the user with the option of entering how many applications
he wishes to run on the grid. Once the number of applications is obtained, the user is
allowed to browse and input these applications.

3. Dividing the applications into sub-jobs.

Once the applications are obtained the scheduler proceeds to divide them into sub-jobs.
The sub-jobs are stored in a queue. The various scheduling algorithms accesses the queue
for scheduling the jobs. While dividing the applications various attributes pertaining to
the sub-jobs are also calculated. Depending upon the lines of input data, the job com-
plexity is found as either low, medium or high. The parameters stored in the queue are as
follows:
Figure 3.6: Architecture of scheduler

- Number of lines of input data
- Job complexity
- Name of the sub job
- Flag variable which states whether the job is mapped or un-mapped

Different parameters are used by different algorithms to assist them in scheduling purposes.

4. Running the scheduling algorithm selected by the user.

Whenever a user provides adequate data about an application, it is divided into a number of jobs. Here we have selected 3 as our division parameter. Scheduler uses an algorithm as selected by the user. This algorithm takes input as the queue of jobs. This queue includes information about the individual jobs. Different algorithms use these parameters differently and generate the appropriate result array. Scheduler first considers the status of the job (i.e., whether it is scheduled or yet to be scheduled) then assigns it to the algorithm selected. The algorithm considers the input and obtains the best possible schedule.

5. Assigning the jobs to the client and obtaining results from them.

This is a stage where the scheduler is ready with schedule in hand. Now scheduler looks
for the grid resources which are to be assigned a job. Scheduler selects a code file and a
data file for each resource. Here we use a technique called as code migration.

**Thread Creation** The Scheduler now checks for the number of Applications that are
added by the User or the Admin sitting at the Server Side. For an application the Sched-
uler creates a thread and for each sub job a new thread which will handle parallel distri-
bution of a schedule is created by the Application thread.

**Code migration** The Application Thread passes the parameters to the Sub Job thread
related to the Grid Resource it has to bind to and to process the Sub Job. The Sub Job
thread reads the Sub Job Data and the Code file and transfers the files to the Grid Resource
using various Java Classes to read the files and to transfer them over the network using
TCP Protocol. The Grid Resource Program running at the Resource receives the Files
and writes them onto the Client. Using the "Process Builder Class" in Java the Client
Complies the code file and further executes the .class file generated. The Output of the
executed file is further a file in which the output is pasted by the code itself. The Client
then sends the output to the Server where the server pastes the file onto its own memory
space. The linking code on the server then uses the output file of each sub-job to integrate
the actual output. The generated output and the total Turn-Around time to solve the
problem is displayed to the user.

6. Rescheduling, in the event of resource failure.

If a job has reportedly failed to execute properly, then it is rescheduled as soon as the
scheduler is informed of its failure. While rescheduling, the Scheduler does not use the
dynamic algorithms. Rather it uses a round robin algorithm for it. The Scheduler gives
the job to the next available resource to the failed resource. The reason for using round
robin algorithm is that since our project is working on LAN, the number of job failures
due to networking will be less.

We are using the TCP protocol for exchange of jobs between the Grid Scheduler and
the resources. Dynamic algorithms can be used for the rescheduling, but their overhead
increases for scheduling a relatively smaller no of jobs. Round Robin does not have to
explore the search space and therefore gives better performance.

7. Merging the results from the various clients an giving it to the user.
Figure 3.7: Usecase model of the scheduler
Figure 3.8: Class model of the scheduler
Once the scheduler gets the results from the various resources, it merge the various outputs together. It then gives the output to the user.

The usecase model of the scheduler shows the interaction among the models of the scheduler. The class diagram gives the outline of the modules with the methods of the classes. The user interface of the scheduler given in shows interaction with use for application submission and selection of algorithm.

### 3.6 Observations

The implementation of the hierarchical scheduler gives benefit of handling applications and decomposition of an application at two levels. Decomposing of an application is the role of local scheduler, so decomposing technique and hence the algorithm for job creation depends on application. Addition of scheduling algorithms with proper interfaces is possible in algorithm selection module hence it is flexible than other schedulers. As the Scheduler also provides the reporting interface for execution time and the computed schedule this will enable user to monitor runtime activities of execution.
Figure 3.9: User interface of the scheduler