Technical Report

Shanghai Jiao Tong University

Report #: SJTU_CS_TR_200309001

A Survey on Grid Scheduling Systems

Yanmin Zhu and Lionel M Ni

A Survey on Grid Scheduling Systems

Yanmin Zhu and Lionel M Ni

Shanghai Jiao Tong University
Hong Kong University of Science and Technology
yzhu@cs.sjtu.edu.cn; ni@cse.ust.hk

Abstract: Thanks to vast improvements in wide-area network performance and powerful yet low-cost computers, Grid computing has emerged as a promising attractive computing paradigm. Computational Grids aim to aggregate the power of heterogeneous, geographically distributed, multiple-domain-spanning computational resources to provide high performance or high-throughput computing. To achieve the promising potentials of computational Grids, an effective and efficient scheduling system is fundamentally important. Scheduling systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Scheduling in Grid environments is significantly complicated by the unique characteristics of Grids. This survey focuses on the design of scheduling systems for computational Grids. First, this survey investigates challenges for designing scheduling system for computational Grids. Second, a Grid scheduling framework and a common Grid scheduler architecture are proposed. Next, a comprehensive taxonomy for describing Grid scheduling systems is also presented. Finally, a number of representative Grid scheduling systems are surveyed in detail.

Keywords: Grid computing, Computational Grid, Parallel Computing, Cluster Computing, Information Collection, Grid Scheduling, Taxonomy

1 Introduction

Thanks to vast improvements in wide-area network performance and powerful yet low-cost computers, Grid computing has emerged as a promising attractive computing paradigm. Computational Grids aim to aggregate the power of heterogeneous, geographically distributed, multiple-domain-spanning computational resources to provide high performance or high-throughput computing. To achieve the promising potentials of computational Grids, an effective and efficient scheduling system is fundamentally important.

Scheduling systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Scheduling in Grid environments is significantly complicated by the heterogeneous and dynamics nature of Grids. Compared to traditional scheduling systems for distributed environments, such as clustering computing, Grid scheduling systems have to take into account diverse characteristics of both various Grid applications and various Grid resources. The different performance goals also place great impacts on the design of scheduling systems. To overcome the heterogeneous and dynamic nature of Grids, the information service plays a highly important role. A successful scheduling system should accommodate all these issues.

This survey focuses on the design of scheduling systems for computational Grids. First of all, challenges for designing scheduling systems in Grid environments are investigated. Then, we propose a Grid scheduling framework which is useful for guiding the design. A common Grid scheduler architecture is discussed with the aim at having insight into possible components for Grid scheduling. A set of taxonomies describing Grid scheduling systems is presented. Finally, a group of representative Grid scheduling systems are surveyed in detail.

1.1 Grid Computing

The term Grid comes from an analogy to a power Grid. When you plug an appliance into a receptacle, you expect that you will be supplied with electricity of correct voltage, whereas you needn't care where the power comes from and how it is generated. In mid-1990s, inspired by the pervasiveness, reliability, and ease of use of electricity, Foster et al [5] began exploring the design and development of an analogous computational power Grid for wide-area parallel and distributed computing.

Since Grid computing is still emerging, the concept of Grid computing itself is evolving. The definition given by Foster [1],[5],[10] is widely accepted and frequently referred. According to Foster, Grid computing strives to aggregate diverse, heterogeneous, geographically distributed and multiple-domain-spanning resources to provide a platform for transparent, secure, coordinated, and high-performance resource-sharing and problem solving. The resources that Grid computing is attempting to integrate are various. They include supercomputers, workstations, databases, storages, networks, software, special instruments, advanced display devices, and even people.

Based on Web Service, the Open Grid Services Architecture (OGSA) [10] [26] and its associated implementation, the Globus toolkit [4] [25], are becoming the de facto standard of Grid service and Grid environment for application development, respectively.

Grid computing is a promising paradigm with the following potential advantages..

- **Exploiting underutilized resources**

Studies have shown that most low-end machines (PCs and workstations) are often idle: utilization is as low as 20% [68]. And even for servers only 50% of their capacity is utilized [68]. Grid computing provides a platform to exploit these underutilized resources and thus has the possibility of increasing the efficiency of resource usage. A simple case is that we can run a local job on a remote machine elsewhere in the Grid if the local machine is busy.

- **Distributed supercomputing capability**

The parallel execution of parallel applications is one of the most attractive features of computational Grids. A wide spectrum of applications is parallel in nature and these applications are intended to be computation-intensive. In Grid systems, there are a large number of computational resources available for one parallel application, such that different jobs within the application can be executed simultaneously on a suite of Grid resources.

- **Virtual organizations for collaboration**

Another important contribution of Grid computing is to enable the collaboration among wider-area members. Grid computing provides the infrastructure to integrate heterogeneous systems to form a virtual organization. Under the virtual organization, sharing is not limited to computational resources, but also includes various resources, such as storages, software, databases, special equipments, and so on. Furthermore, the sharing is more direct through using the uniform interfaces. Although sharing in a virtual organization is quite direct, security and local policy are guaranteed. Local resources are protected securely against those who are not authorized to access.

- **Resource balancing**

After joining a Grid, users will have a dramatically larger pool of resources available for their applications. When the local system is busy with a heavy load, part of the workloads can be scheduled to other resources in the Grid. Thus the function of resource balancing is achieved. This feature proves to be invaluable for handling occasional peak loads on a single system.

- **Reliability**

High-end conventional computing systems use expensive hardware to increase reliability. In the future, Grid computing provides a complementary approach to achieving high-reliability nevertheless with little additional investment. The resources in a Grid can be relatively inexpensive, autonomous and geographically dispersed. Thus, even if some of the resources within a Grid encounter a severe disaster, the other parts of the Grid are unlikely to be affected and remain working well.

1.2 Grid System Taxonomy

Grid computing is still a very general concept. Different institutes or organizations devise different Grid systems to meet their specific needs. Grid computing can be used in a variety of ways to address various kinds of application requirements.

According to the distinct targeted application realms, Grid systems can be classified into three categories. But there are actually no hard boundaries between these Grid categories. Real Grids may be a combination of two or more of these types. The three categories of Grid systems are described below.

- **Computational Grid**

A computational Grid is a system that aims at achieving higher aggregate computational power than any single constituent machine. According to how the computing power is utilized, computational Grids can be further subdivided into *distributed supercomputing* and *high throughput* categories. A distributed supercomputing Grid exploits the parallel execution of applications over multiple machines simultaneously to reduce the execution time. A high throughput Grid aims to increase the completion rate of a stream of jobs through utilizing available idle computing cycles as many as possible.

- **Data Grid**

A data Grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life science research, each with unique data. A data Grid would allow them to share their data, manage the data, and manage security issues. European DataGrid [41] Project is one example of Data Grids.

- **Storage Grid**

A storage Grid attempts to aggregate the spare storage resources in Grid environments and provides users transparent and secure storage services.

On the other hand, Grids can be built in all sizes, ranging from just a few machines in a department to groups of machines organized as hierarchy spanning the world. Grids can be classified into three categories according to the topology of Grid. The relationship between the three Grid topologies is illustrated in Figure 1-1.

- **IntraGrid**

A typical intraGrid topology exists within a single organization. The single organization could be made up of a number of computers that share a common security domain, which are connected by a private high-speed local network. The primary characteristics of an intraGrid are a single administrative domain and bandwidth guarantee on the private network. Within an intraGrid, it is easier to design the scheduling system, since an intraGrid provides a relatively static set of computing resources and communication capability between machines.

- **ExtraGrid**

An extraGrid couples two or more IntaGrids. The extraGrid typically involves more than one administrative domains, and the level of management complexity increases. The primary characteristics of an ExtraGrid are dispersed security, multiple domains, and remote/WAN connectivity. Within an ExtraGrid, the resources become more dynamic. A business would benefit from an ExtraGrid if there was a business initiative of integrating with external trusted business partners.

- **InterGrid**

An InterGrid has an analogy with the Internet. It is the most complicated form of Grid topology. The primary characteristics of an interGrid include dispersed security, multiple domains and WAN connectivity. A business may deem an InterGrid necessary if there is a need for a collaborative computing community, or simplified end to end processes with the organizations that will use the interGrid.

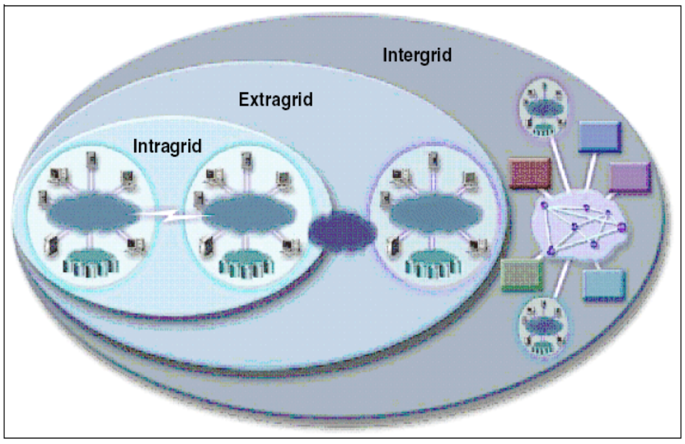


Figure 1-1: IntraGrid, ExtraGrid, and InterGrid [2]

Both the targeted application realms and Grid topology will fundamentally impact the design and development of Grid systems. In this survey, we speak of scheduling in the context of computational Grids since we will concentrate on the problem of scheduling user applications for execution on a suite of computational resources.

1.3 Terminologies

The scheduling problem in parallel environments has been an active research topic, and therefore many terminologies have been suggested. Unfortunately, some of the terms are neither clearly stated nor consistently used by different researches, which frequently makes readers confused. For clear discussion in this survey, some key terminologies are re-defined.

Application and Job

Grid application: A Grid application is a collection of jobs coordinated to solve a certain problem. In other words, a Grid application may consist of a number of jobs, either dependent or independent, that together fulfill the whole task.

Job: A job is considered as a single unit of work within a Grid application. It is typically allocated to execute on one single computational resource in the Grid. It has input and output data, and execution requirements in order to complete its task.

Site

We consider a *site* as a computational resource that can interact with Grid schedulers directly and accept workloads from the schedulers. A site may be a simple personal machine, a workstation, a supercomputer, or a cluster of workstations. From a scheduler's a view of point, a site is an atomic resource unit that can be allocated to application jobs.

Processing node

A processing node (also referred as computing node or computational node) means different things under different environments. Under the context of Grid environments, a processing node is a site. However, in the context of Cluster environments, a processing node means a stand-alone computer.

SISD, SIMD, MISD and MIMD

SISD, SIMD, MISD, and MIMD are the acronyms of Single-Instruction Single-Data, Single-Instruction Multiple-Data, Multiple-Instruction Multiple-Data and Multiple-Instruction Multiple-Data, respectively. According to Flynn's taxonomy [73], computer architectures can be classified into these four categories in terms of the control flows and data flows.

- SISD: a single instruction flow operates on a single data flow. This describes the sequential processing techniques.
- SIMD: a single instruction flow operates on multiple data flows simultaneously.

- MISD: multiple instruction flows operate on single data flow simultaneously. It seldom appears in real architectures.
- MIMD: multiple instruction flows operate on multiple data flows simultaneously.

SIMD and MIMD are two widely-used classes of parallel computing architectures.

1.4 Problem Formulation

A scheduler is the mediate resource manager as the interface between the consumers and the underlying resources, as illustrated in Figure 1-2. Scheduling is a core function of resource management systems.

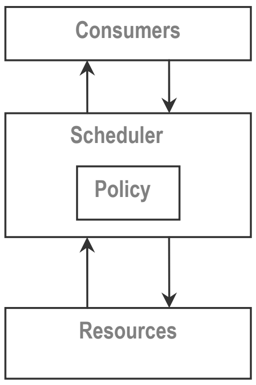


Figure 1-2: Scheduler Function

In a distributed environment, on one hand, there is a suite of computational resources interconnected by networks; on the other hand, there is a group of users who will submit applications for execution on the suite of resources. The scheduling system of such a distributed computing environment is responsible for managing the suite of resources and dealing with the set of applications. In face of a set of applications waiting of execution, the scheduling system should be able to allocate appropriate resources to applications, attempting to achieve some performance goals.

In traditional parallel computing environments, the scheduling system is made much simpler due to the uniform characteristics of both the target applications and the underlying resources. However, a computational Grid has more diverse resources as well as more diverse applications.

According to GGF's Grid scheduling dictionary [27], the Grid scheduler is responsible for:

(1) Discovering available resources for an application

(2) Selecting the appropriate system(s), and

(3) Submitting the application.

In brief**, Grid scheduling** is a software framework with which the scheduler collects resource state information, selects appropriate resources, predicts the potential performance for each candidate schedule, and determines the best schedule for the applications to be executed on a Grid system subject to some performance goals.

In principle, scheduling in Grids means two things: ordering and mapping. When there are more than one applications waiting for execution, *ordering* is performed in order to determine by which order the pending applications are arranged. *Ordering* is necessary if applications with priority or deadline are involved. *Mapping* is the process of selecting a set of appropriate resources and allocating the set of resources to the applications. For each mapping, the performance potential is estimated in order to decide the best schedule.

In general, a scheduling system of Grid computing environments aims at delivering better performance. Desirable performance goals of Grid scheduling includes: maximizing system throughput [35], maximizing resource utilization, minimizing the execution time [21] and fulfilling economical constraints [20].

1.5 Organization of the Paper

The remaining part of the survey is organized as follows. Section 2 reviews the scheduling systems for traditional distributed-memory parallel environments (in this survey, clusters are selected for discussion). In Section 3, first the challenges for Grid scheduling are investigated. Second, a Grid scheduling framework and a common Grid scheduler architecture are proposed. Finally, the procedure and the strategies of Grid scheduling are discussed. Section 4 provides a comprehensive taxonomy of Grid scheduling systems. In Section 5, a group of representative existing scheduling systems for Grids is discussed in detail. I summarize and discuss future work in Section 6.

2 Traditional Parallel Scheduling Systems

Before going deeper into Grid scheduling, the scheduling systems of traditional parallel computing should be examined. Because of its high importance, the scheduling problem for parallel systems has been the research topic of a large body of work.

2.1 Traditional Parallel Systems

Traditionally, parallel computing systems can be classified into two categories: parallel shared-memory supercomputers and clusters of workstations. In recent years, more and more expensive parallel supercomputers are being replaced by clusters. Clusters can provider similar or even better performance compared to parallel supercomputers, while clusters cost much less. Clusters have become the low-cost and standard platforms for parallel computing.

A cluster is a collection of stand-alone computing nodes (usually low-end systems, such as PCs) which are interconnected by a high-speed system area network, and work together as a single integrated computing system [37]. Such a cluster is administrated by a single entity which has complete control over all the computing nodes. One popular implementation is a cluster with nodes running Linux as the OS and Beowulf software (both free software) to support the parallelism. The common architecture of clusters is shown in Figure 2-1.

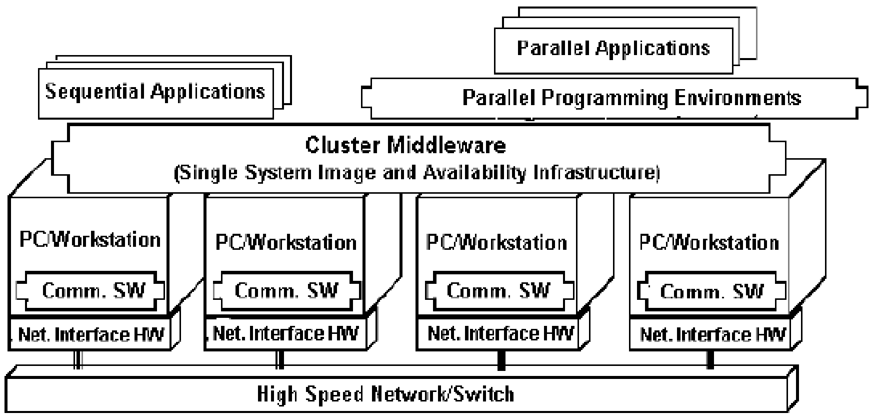


Figure 2-1: Typical Cluster Architecture [37]

Thus in this survey, we consider a cluster as the representative of traditional parallel computing systems. Cluster scheduling is discussed in the remaining part of this section.

Clusters are organized into a master-slave model. The master node is responsible for accepting jobs and scheduling these jobs onto the slave nodes to execute. A slave node is a stand-alone computer, which has its own CPU, memory, and operating system. The slave nodes are responsible for executing jobs and returning the results to the users. The slave nodes are usually interconnected by a high-speed network allowing low-latency, high-bandwidth inter-processor communications.

Typically, inter-processor communication is implemented by message-passing mechanisms, such as PVM [69] and MPI [70]. A cluster is suitable to run both sequential jobs and parallel jobs. The parallelism of a parallel application is designed by programmers based on message passing mechanisms. The parallel execution on a cluster requires the rum-time library support of PVM or MPI.

2.2 Cluster Scheduling

As shown in Figure 2-2, the cluster scheduling system has a master-slave architecture. The master node receives jobs from clients, and puts them in a queue first. Hence the master node has the overall information of all the submitted jobs. It is the master node's responsibility to schedule the jobs waiting in the queue onto to the slave nodes for execution. Hence the master node acts as the scheduler as well.

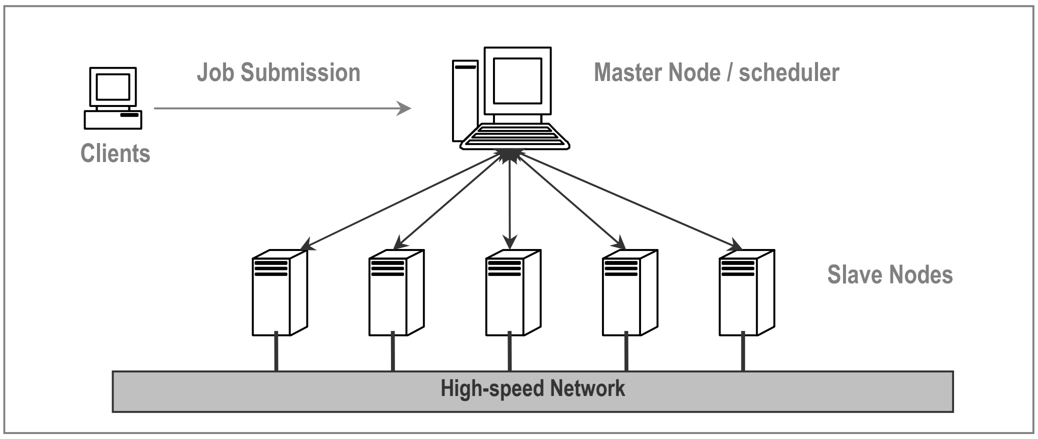


Figure 2-2: Cluster Scheduling Architecture

In cluster computing, all slave nodes are typically uniform in terms of CPU speed, memory size, and network bandwidth. And the set of slave nodes is usually fixed. Every node in the cluster is dedicated to cluster computing. The underlying interconnection network is private and high-speed, such that the communication between every pair of nodes can be guaranteed in most cases. The master node is fully in charge of the fixed set of slave nodes which can be allocated to jobs. When making scheduling decision, the scheduler concerns the availability of each slave node only. Once a slave node completes executing a job, it reports its availability to the scheduler. Hence at any time the scheduler knows the overall information of the fixed set of slave nodes.

In general, each job specifies its resource requests in a script file which the scheduler will interpret. A parallel job may consist of several subjobs, each of which is expected to run on a separate slave node. Resource requests of a job include the number of nodes and expected running time. Such resource requirements are typically provided by the programmers.

The scheduler may perform ordering on the pending jobs in the queue. The algorithms used to guide the ordering operation vary from cluster to cluster. The algorithms frequently used include: First-Come-First-Serve, Minimal-Request-Job-Fist, Shortest-Job-First, Backfill [37] and so on. After ordering, the job waiting in head of the queue is considered to schedule next. Based on the job's resource request, the scheduler selects the requested number of slave nodes for the job, and later allocates the job onto the set of selected slave nodes. After all the subjobs complete their execution, the result of the job is written into a file on the master node and later this file is sent to the client who submitted the job.

The performance goals of the scheduling systems for clusters are usually system-centric. Desired cluster performance goals include maximizing throughput and maximizing resource utilization.

2.3 Characteristics of Cluster Scheduling

The characteristics of cluster scheduling highly result from the characteristics of cluster computing environments. The characteristics of cluster environments and cluster scheduling methods are discussed concisely in the following:

Homogeneity of resources and applications: Because of the homogeneity of computing nodes, the estimation of execution time of a job on a computing node is greatly simplified. Since a cluster has a similar group of users, applications from these users have similar structures and resources requirement models.

Dedicated resources: Most clusters only span one single administrative domain. Both the computing nodes and the interconnection network in a cluster are dedicated to the purpose of cluster computing. Hence the behavior of both the computing nodes and the communication capacity are easy to predict. Some researchers have worked on scheduling in non-dedicated workstation clusters. Each node may share the spare cycles with others while the node has an individual owner.

Centralized scheduling architecture: In a cluster, the function of scheduling is performed by centralized entity in the cluster. The centralized scheduler has a complete control over all computing nodes and the overall information about the pending jobs. Thus the scheduler can schedule jobs onto the cluster effectively and efficiently.

High-speed interconnection network: Due to the sufficient bandwidth provided by high-speed private interconnection network, the local of computing nodes are not critical.

Monotonic performance goal: There is one single performance goal for the whole cluster. The operations of scheduling are oriented to approach the performance goal. The scheduling design is simplified due to the monotonic performance goal.

In summary, scheduling systems for cluster environments can provide a basis, but not a complete solution to the scheduling problem for Grid systems. However, cluster computing management software including the scheduling function will necessarily be part of the local resource management solutions. Grid scheduling systems should be able to cooperate with these local cluster schedulers to form a wider scheduling system. In other words, a point of a grid could be a cluster.

3 Grid Scheduling: Challenges, Framework and Architecture

The initial approaches to designing Grid scheduling systems are based on those methodologies for cluster scheduling. But the assumptions made in cluster environments do not hold in Grid environments any longer. Computational Grids have much in common with clusters, both of which are distributed parallel computing environments, yet differ significantly from clusters in many important ways. Consequently, poor performance will be experienced if we apply cluster scheduling to Grid computing environments.

Before we attempt to develop Grid scheduling systems, the challenges posed by Grid environments should be carefully examined. Later a scheduling framework is proposed which is believed to be useful for guiding the design of Grid scheduling. A common architecture of Grid scheduler is scratched at the end of this sector.

3.1 Challenges for Grid Scheduling

Although a Grid also falls into the category of distributed parallel computing environments, it has a lot of unique characteristics which make the scheduling in Grid environments highly difficult. An adequate Grid scheduling system should overcome these challenges to leverage the promising potential of Grid systems, providing high-performance services.

The grand challenges imposed by Grid systems are examined in detail in the following:

- **Resource Heterogeneity**

A computational Grid mainly has two categories of resources: networks and computational resources. Heterogeneity exists in both of the two categories of resources. First, networks used to interconnect these computational resources may differ significantly in terms of their bandwidth and communication protocols. A wide-area Grid may have to utilize the best-effort services provided by the Internet. Second, computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architectures, number of processor, physical memory size, CPU speed, and so on, and also different software, such as different operating systems, file systems, cluster management software, and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity could not be considered uniformly. An adequate scheduling system should address the heterogeneity and further leverage different computing power of diverse resources.

- **Site autonomy**

Typically a Grid may comprise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the domain. Thus applications from non-authorized users should not be eligible to run on the resources in some specific domains.

Further more, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own scheduling policy, which complicates the prediction of a job on the site. A single overall *Performance goal* is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

Local priority is another important issue. Each site within the Grid has its own scheduling policy. Certain classes of jobs have higher priority only on certain specific resources. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources.

Most traditional schedulers are designed with the assumption of having complete control of the underlying resources. Under this assumption, the scheduler has adequate information of resources and therefore effective scheduler is much easier to obtain. But in Grid environments, the Grid scheduler has only limited control over the resources. Site autonomy greatly complicates the design of effective Grid scheduling.

- **Resource Non-dedication**

Because of non-dedication of resources, resource usage c*ontention* is a major issue. Competition may exist in both computational resources and interconnection networks. Due to the non-dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well. One consequence of contention is that behavior and performance can vary over time. For example, in wide area networks using the Internet Protocol suite, network characteristics such as latency and bandwidth may be varying over time. Under such an environment, designing an accurate performance model is extremely difficult.

Contention is addressed by assessing the fraction of available resources dynamically, and using this information to predict the fraction available at the time of application to be scheduled. With quality-of-service (QoS) guarantees and resource reservation provided by the underlying resource management system, predicting resource performance is made easier. A scheduler can regard the fraction of those resources that are protected by OoS guarantees as "dedicated" (contention-free) at the guaranteed level. Schedulers must be able to consider the effects of contention and predict the available resource capabilities.

- **Application diversity**

The problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs, and others may consist of a set of dependent jobs. In this context, building a general-purpose scheduling system seems extremely difficult. An adequate scheduling system should be able to handle a variety of applications.

- **Dynamic behavior**

In traditional parallel computing environments, such as a cluster, the pool of resources is assumed to be fixed or stable. In a Grid environment, dynamics exists in both the networks and computational resources. First, a network shared by many parities cannot provide guaranteed bandwidth. This is particularly true when wide-area networks such as the Internet are involved. Second, both the availability and capability of computational resources will exhibit dynamic behavior. On one hand new resources may join the Grid, and on the other hand, some resources may become unavailable due do problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources. An adequate scheduler should adapt to such dynamic behavior. After a new resource joins the Grid, the scheduler should be able to detect it automatically and leverage the new resource in the later scheduling decision making. When a computational resource becomes unavailable resulting from an unexceptional failure, mechanisms, such as checkpointing or rescheduling, should be taken to guarantee the reliability of Grid systems.

These challenges pose significant obstacles on the problem of designing an efficient and effective scheduling system for Grid environments. Some problems resulting from the above are not solved successfully yet and still open research issues. As a result, new scheduling frame work must be developed for Grids, which should reflect the unique characteristics of Grid systems.

3.2 Grid Scheduling Framework

A complete Grid scheduling framework comprises *application model*, *resource model*, *performance model, and* *scheduling policy*. ***An application model*** extracts the characteristics of applications to be scheduled. ***A resource model*** describes the characteristics of the underlying resources in Grid systems. ***A performance model*** is responsible for predicting the performance potential of a schedule. The prediction is usually based on the prediction of the behavior of a specific job on a specific computational resource. **S*cheduling policy*** is responsible for deciding how applications should be executed and how resources should be utilized.

The effectiveness of Grid scheduling systems is highly based on the development of an adequate scheduling framework for computational Grids. Developing an efficient scheduling framework is difficult. The hardship results from both the heterogeneity of Grid resources and the diversity of Grid applications.

3.2.1 Grid Application Model

The Grid application model is used to describe the characteristics of Grid applications. The model should be able to parameterize a user application to form a description of the application as the input to the performance model. Characteristics of applications can be viewed from many aspects. In the following, let's examine some of them.

Application flow

If you want to take advantage of parallel execution, you must determine whether the application can be executed in a parallel way. An application flow reflects the inter-job precedence.

An application flow is the flow of work among its constituent jobs. Through determining the application flow, we can better allocate the application on a Grid environment for execution. There are three basic types of application flow that can be identified.

- Parallel flow

In this case, there is an initial job, followed a number of parallel jobs. And finally an ending job is responsible for collecting the results of each job. Each job in the set of parallels jobs may receive a discrete set of data, and fulfills its computational task independently and delivers its output. Parametric study [21] is a case of parallel flow application. Applications with parallel flows are well suited for deployment on a Grid. Data-parallel applications have parallel flow.

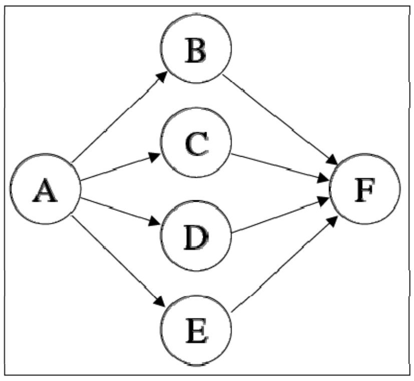


Figure 3-1: Parallel Flow [2]

- Serial flow

In contrast to the parallel flow, a serial application flow has a single thread of job execution where each of the subsequent jobs has to wait for its predecessor to end and deliver output data as input to the next job. This means any job is a consumer of its predecessor, the data producer. In this case, the advantages of running in a Grid environment are not based on access to multiple systems in parallel, but rather on the ability to use any of several appropriate and available resources.

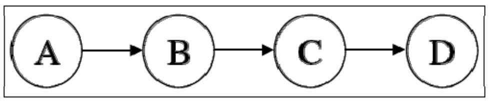


Figure 3-2: Serial Flow [2]

- Networked flow

The networked flow is the type that we encounter in really applications. As shown in Figure 3-3, certain jobs within the application are executable in parallel, but there are interdependences between them. In the example, jobs B and C can be launched simultaneously, but they heavily exchange data with each other. Job F cannot be launched before B and C have completed, whereas job E or D can be launched upon completion of B or C, respectively. Finally, job G collects all output from the jobs D, E, and F, and its termination and results then represent the completion of the Grid application.

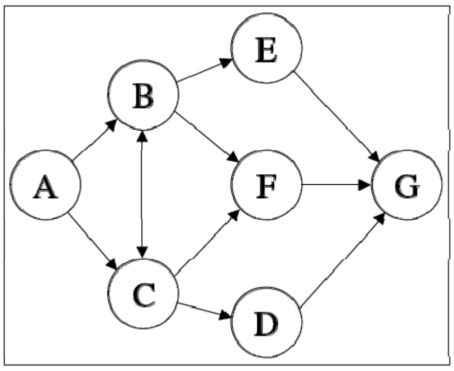


Figure 3-3: Networked Flow [2]

Job flow

As discussed above, a job is the unit of work that will be allocated to a site. A job may further have a work flow, called job flow, in which there exists another level of parallelisms. The job flow reflects how a single computational site processes the job. Parallelism in a single job is much finer grained than that of an application. In principle, the parallelism in a job should be utilized by the local computational resources. For example, an SIMD supercomputer is suitable to execute jobs with SIMD type of parallelisms. This will be discussed in more detail in the following.

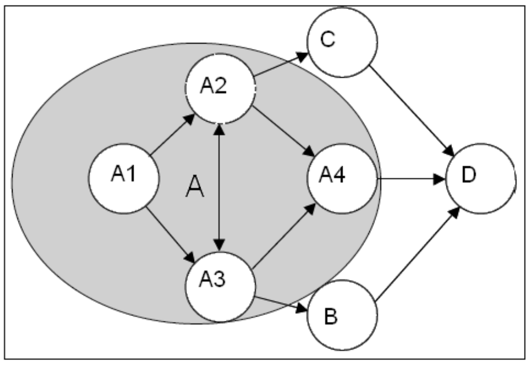


Figure: 3-4 Job Flow [2]

Level of parallelism

Parallel executing is an important approach to reducing the execution time of computation. Careful studies show that parallelism exists in different levels. Let's take a closer look at the parallelism level.

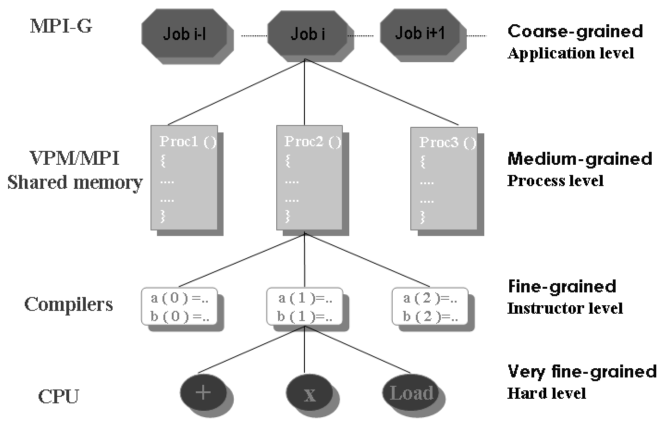


Figure 3-5: Parallelism Levels [37]

As shown in Figure 3-5, there are possibly four levels of parallelisms. First, we speak of application-level parallelism if an application consists of a set of parallel jobs. This level of parallelisms is so-called coarse-grained and implemented in the context of Grid scheduling. The parallelisms of application level are usually designed by application developer. Second, we speak of process-level parallelism if a job consists of a set of parallel processes. Process-level parallelisms are medium-grained and implemented by resources manage software, such as operating systems and cluster management software (e.g. PBS). This level of parallelisms is usually designed by programmers. Finally, there are still two lower levels of parallelisms: Instructor-level and hardware-level parallelism. We do not concern the two lower of parallelism in the context of Grid scheduling.

Parallel Application's Representation

Generally, A complex parallel application can be represented by a directed acyclic graph (DAG) G=(V,E) where V is a set of v=|V| nodes and E is a set of e=|E| directed edges. A node in the DAG represents a job which will be executed in the one processing node. The weight of a node is called the computation cost. The edges in the DAG correspond to the communication and precedence constraints among the nodes. The weight of an edge is called the communication cost of the edge. The precedence constraints of a DAG dictate that a node cannot start execution before it gathers all of the data from its preceding nodes. The communication cost between two jobs assigned to the same processor is assumed to be zero.

Applications Classification

1. Batch vs. Interactive

Applications can be classified into two categories according to whether the jobs will require further interaction with users after submission for execution.

- Batch

A batch application is submitted for execution and is processed without any further interaction from the User. In general, a batch application is well suited for deployment in a Grid environment.

- Interactive

An interactive application needs user's input after which the application can continue its execution. There would be many considerations and issues involved in the development and deployment of such interactive job within a Grid environment.

2. Real-time vs. non real-time

Applications are classified into two categories: realtime and non real-time, according to whether the application specifies a deadline by which the application must complete its execution.

- Real-time

All real-time applications define a deadline by which they have to complete executing. When real-time applications are involved in scheduling, the performance goal is usually the completion before the predefined deadline. In some real-time systems, if an application could not be completed before its deadline, severe results would follow. This is called a hard real-time application.

- Non Real-time

In general, a non real-time application does not care the deadline. Users with such applications usually submit applications to the Grid system, and get the results back at a later time. They may more concern other performances, such as cost.

3. Priority

Priority is a common technique in scheduling. A set of applications assigned with priorities will be scheduled based on not only the objective performance but also the relative weighting of their priorities. In a preemptive system, an application with a low priority may be preempted from execution by another application with a higher priority.

Application description granularity

The application description will be used as the input to the performance model. Thus the granularity of the application description partially determines the quality of performance prediction of the application. The description of an application and its constituent jobs can either fine-grained or coarse-grained. A coarse-grained application description is described in a high level, and provides little information of the application. Whereas, a fine-grained application description lists the detailed information of the application and its constituent jobs. For example, a fine-grained application description may include the job dependency, the data size that should be transfer from one node to anther, the parallelism type of each job, and so on. Much detailed information of applications helps to obtain much better matches with the underlying resources, and hence more accurate performance predictions.

3.2.2 Grid Resource Model

The Grid resource model is responsible for specifying the characteristics of Grid resources. Resources in a Grid environment may be dramatically heterogeneous. They may differ from each other in many aspects, such as whether they are dedicated to a Grid, whether a resource is time shared, etc.

For a computational resource, we concern its capability. We used to measuring its capability by its CPU speed. But in effect, the capability of a computational resource varies relatively with respect to different kinds of jobs. The capability of a computational resource for a specific job depends on many things: CPU speed, memory size, the degree of match between the parallelism type supported by the resource and the parallelism type that one job has.

Supported Parallelism Type

Every kind of computational resource may be suitable for a class of parallelism types. As pointed out previously, there are needs on a match between the parallelism type requested by a job and the parallelism type supported by the computational resource. If the match is perfect, the job's performance would be optimized. Otherwise, the job's performance would be decreased. Different parallel machines have been designed, such as SIMD, MIMD, vector computers, pipelining systems, and so on, each of which is corresponding to one parallelism type exhibited by a job.

Resource Classification

1. Time-shared vs. *non- time-shared*

In a time-shared resource, multiple jobs are running concurrently, each executing for a time-slice in turn. A time-shared resource delivers differing performance with respect to the current workload on the resource. It is relatively difficult to make a performance prediction for a job on such a computational resource. When time-shared resources are involved, we would like to determine the capability available to a given job, such as memory size, percentage of available CPU, etc.

In a non-time-shared resource, one job is executed at a time. That means the computational resource receives one job only at a time, and processes jobs one by one. Alternatively the computational resource queues job locally and executes jobs one by one. It is relatively easier to make a performance prediction for a job on such a computational resource since only one job is running on the resource at a time. When a non-time-shared resource is involved, we prefer determining the time that it will be available to a given job.

2. Dedicated vs. non-dedicated

Depending on the ownership or its purpose, Grid resources can be divided into two categories: dedicated and non-dedicated resources. A dedicated resource is fully deployed for purpose of use in a specific Grid. A dedicated receives workloads only from a single Grid.

In contrast, a non-dedicated resource may participate in multiple Grids at the same time. Thus a non-dedicated resource will receive workloads from multiple Grids as well as local users. Due to the potential contention, the state information of these resources, such as CPU workloads, available memory sizes and bandwidth of network, is varying over time, and hence accurate performance predicting for non-dedicated resources is a hard issue. In a Grid environment, non-dedicated resources are targeted to leverage the abundant computing cycles available to provide high computing power without additional financial investment. These resources have their own workloads.

3. Preemptive vs. non-preemptive:

In a preemptive computational resource, a running job can be preempted from execution. Preemption is useful in priority-based or real-time systems. For example, when a pending job's deadline is approaching, it is necessary to preempt one running job whose deadline is much looser and assign the resource to that job. As a result of preemption, the scheduling is much complicated.

A non-preemptive computational resource does not allow preemption, i.e., once a job is started on such a resource, the job cannot be preempted until its completion.

Resource description granularity

In general, the resource information is an important part of the input data to the performance model. Similar with the situation for application descriptions, the description of a Grid resource can be either fine-grained or coarse-grained. A coarse-grained description may tell the OS type, hardware architecture only. But a fine-grained resource description lists all detailed information related to the resource. For example, a fine-grained resource description may include the CPU speed, the number of CPUs， memory size, parallelism type supported, and so on. Clearly, a fine-grained resources description is of much help to the performance prediction.

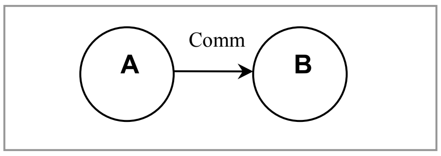
It should be noticed that the accurate descriptions may not be provided by the resources themselves, Instead, the information service infrastructure in a Grid environment is widely exploited to provide such information.

3.2.3 Performance Model

The performance model the most complicated part of a scheduling framework. Basically, a performance model takes into account the application description and the resource description, and estimates the performance potential for each schedule, according to the performance goal defined in the scheduling policy. A performance model seeks to obtain as accurate as possible performance prediction for each schedule of a set of jobs mapping to a set of resources. A performance model is usually application-specific, meaning that it is desired that the performance knows some knowledge about the applications such as typical application flows.

The Grid scheduler selects a set of feasible resources for a set of jobs and forms a list of candidate schedules. The performance potential of each schedule is measured according to the performance model. The quality of performance prediction provided by performance model has a significant impact on the effectiveness of the scheduler. Grid performance models must be sufficiently complex to represent the phenomena of both the applications and the resources.

For example, one class of applications consists of two jobs: A and B. A must complete before it communicates with B. A performance model is established to predict the execution time of the application.



Under the performance model, the execution time can be expressed as follows:

ExecTime=CompTime(A) + CommTime(A-B) + CompTime(B)

Where **CompTime(A)** and **CompTime(B)** represent the estimated computation time of job A and B on some computational resources, respectively, and **CommTime(A-B)** provides the estimated communication time between job A and B over some communication channel.

Dynamic Information

In Grid environments, computation time may depend upon CPU load, and communication time may depend upon available network bandwidth. Thus a more accurate performance model should leverage this dynamic information to provide much closer estimation compared to the real performance.

Adaptation

A performance model can be adapted to reflect the characteristics of applications and execution environments. For example, the performance model described above may not work well if that application allows overlapped communication and computation. An adequate performance model should be adaptive and reflect the particular characteristic of the application. If the communication and computation could be overlapped, the following estimation would be much appropriate:

ExecTime=MAX{ CompTime(A), CommTime(A-B), CompTime(B) }

Adaptation can also be used to weight the relative impact of the performance activities represented in a performance model. For example, if our example application is compute-intensive, it may be very important to derive an accurate model for CompTime(A) and CompTime(B), and less important to derive an accurate model application execution time.

3.2.4 Scheduling policy

The scheduling policy determines how an application should be scheduled and how the resources should be utilized. Most importantly, the scheduling policy is responsible for defining the performance goals for the Grid system. Based on the performance model, the performance potential of each candidate schedule is computed. According to the performance goal, the schedule which will produce the best performance potential is selected. Other responsibility of scheduling policy may include the constraint that local jobs should have high priority of using local resources. The scheduling policy has relatively direct impact on the design of the performance model.

As for the performance goals, two different classes are commonly targeted. First, Grid users concern the performance of applications. Second, Grid administrator or resource owner may concentrate on the overall utilization of the whole system. We'd like to term these two conflicting classes of performance goals: application-centric and system-centric, respectively.

In the following, some common performance goals in both classes are listed:

1. *application-centric*

An application-centric scheduling policy seeks to optimize the performance of each individual application.

- Execution time: the time duration spent executing the job.
- Waiting time💻 ♦♒♏ ♦♓❍♏ ♎◆❒♋♦♓□■ ⬧□♏■♦ ⬧♋♓♦♓■♑ ♓■ ♦♒♏ ❒♏♋♎⍓ ❑◆♏◆♏
- Speedup: the ratio of time spent executing the job on the original platform to time spent executing the job on the Grid.

- Turnaround time: also called response time. It is defined as the sum of waiting time and execution time.
- Job slowdown: it is defined as the ratio of the response time of a job to its actual run time.

2. System-centric

- Throughput:
  ⧫♒♏ ■◆❍♌♏❒ □➢ ☺□♌⬧ ♍□❍◻●♏⧫♏♎ ♓■ □■♏ ◆■♓⧫ □➢ ⧫♓❍♏📪 ⬧◆♍♒ ♋⬧ ◻♏❒ ♒□◆❒ □❒ ◻♏❒ ♎♋⍓📪
- Utilization: ⧫♒♏ ◻♏❒♍♏■⧫ □➢ ⧫♓❍♏ ♋ ❒♏⬧□◆❒♍♏ ♓⬧ ♌◆⬧⍓
- Makespan: the makespan of a set of jobs is defined as the spanning time before the completion of the last job.
- Flow time: the flow time of a set of jobs is the sum of completion time of all jobs.
- Average application performance.

3.3 A Common Grid Scheduler Architecture

In this subsection, a common Grid scheduler architecture is given. The functionalities of the components that comprise the overall Grid architecture are explained. Also, the interactions between these components are discussed.

Before presenting the Grid scheduler architecture, it is beneficial to investigate the physical architecture of a computational Grid as shown in Figure 3-6. A computational Grid typically spans multiple administrative domains. Every domain has its own administrative and security policy. A domain may have one or several sites which are typically connected by local high-speed network. A site may be a stand-alone workstation, a supercomputer, or even a cluster of workstations controlled by some cluster management software. The interconnection network is used to interconnect all the resource from every domain. The interconnection network may be a wide-area network, such as the Internet, or a dedicated private connection link. Shared wide-area network provides varying performance, which is difficult to predicate the communication performance.

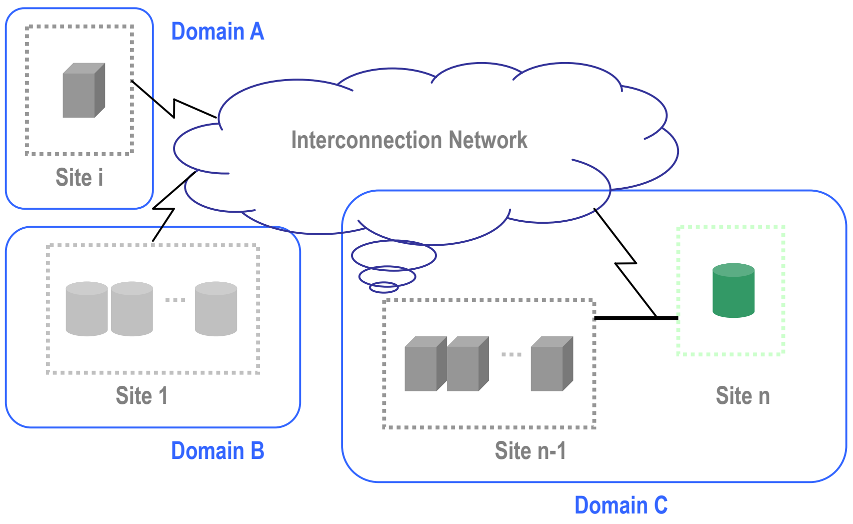


Figure 3-6: Computational Grid Physical Architecture

Figure 3-7 depicts a common architecture of a Grid scheduler.

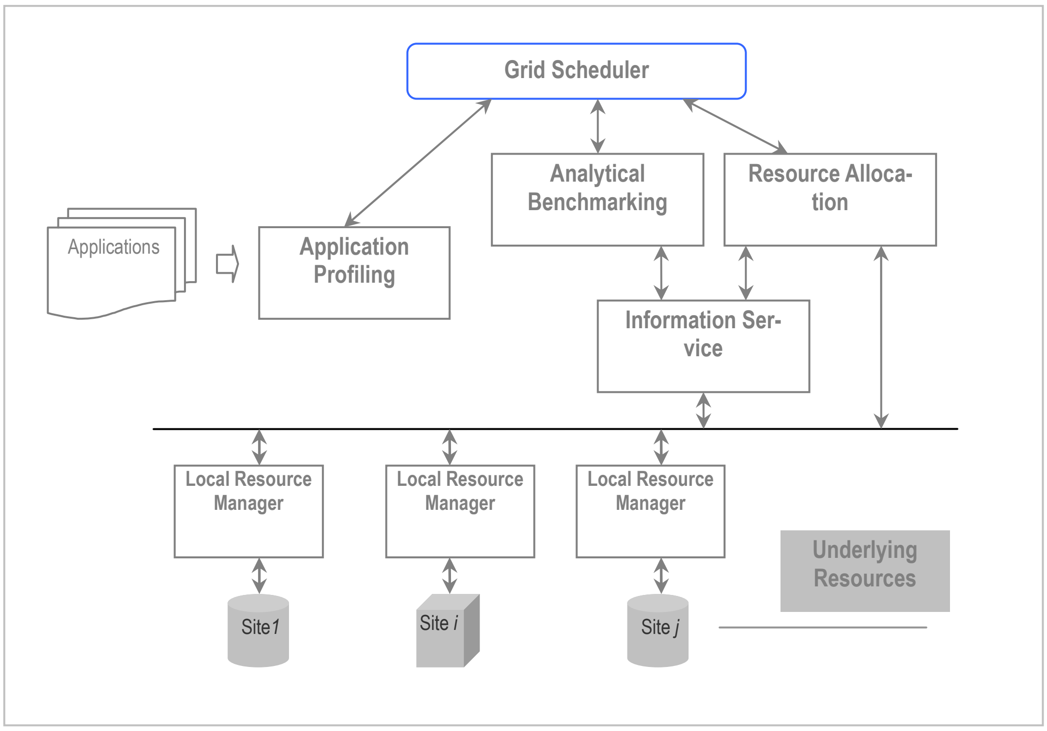


Figure 3-7: A Common Grid Scheduler Architecture

Real resources lie on the bottom of the architecture in the figure, each being managed by a *Local Resource Manager* (**LRM**). An LRM is responsible for:

- processing low-level resource requests, by executing a job that satisfy that request;
- enabling remote monitoring and management of jobs created in response to a resource request; and
- updating the information service with information about the current availability and capabilities of the resources that it manages.

An **LRM** serves as the interface between a Grid scheduler and a local autonomous site being able to process low-level resource request. Grid Resource Allocation Management **(GRAM)** from Globus [4] [25] is a good example of LRM.

Information service (**IS**) component is responsible for maintaining the current state information of resources, such as the CPU capacity, memory size, network bandwidth, software availability, and so on, and answering to queries for acquiring information about Grid resources. To overcome the heterogeneous and dynamics nature of Grid environments, efficient information service plays a particularly important role in the Grid scheduling system. An adequate scheduler must incorporate the current information of resources when making scheduling decisions. Globus Meta Director Service (MDS) [24] and Network Weather Service (NWS) [28] are two mature examples that provide information service.

User applications are fed into the Grid scheduling system through the interface *Application Profiling* (**AP**). As discussed previously, the application description specifying the characteristics of applications is necessary for accurate performance prediction. There are two ways to obtain the application description. One way is to obtain it from the users' specification. The other way is to use the application profiling to extract the characteristics such as the parallelism type and inter-job dependency.

Analytical Benchmarking **(AB)** component provides a measure of how well a computational resource performs on a given job. The execution time of a given job varies with the capability of resources. Thus basically, the **AB** component provides a performance estimation of a given job on a specific computational resource.

Grid Scheduler (**GS**) is the core component in the architecture. The **GS** needs to do two jobs: one is *resource selection* and the other one is *mapping*. Resource selection is the process of selecting feasible and available resources for a given application to be scheduled. Mapping is the process of placing the jobs and communications of the application onto the resources and networks. Each mapping of jobs to feasible resources produces a candidate schedule. Each candidate schedule is then estimated for its performance potential based on the performance model.

Resource Allocation (**RA**) component implements a finally-determined schedule through allocating the resources to the corresponding jobs. Resource allocation may involve data staging and binary code transferring before the job starts execution on the computational resource.

The common Grid scheduler architecture provides a high-level view of Grid schedulers. Not all existing Grid scheduling systems have corresponding components that appear in the architecture. But every component seems necessary for any comprehensive Grid scheduling systems.

3.4 Grid Scheduling Procedure

Through above discussion, it is quite clear that the Grid scheduling procedure roughly comprises four phases: *information collection*, *resources selection*, *job mapping* and *resource allocation*.

1. Information Collection

Information Collection is the basis for providing current state information of the resources. It should be performed during the whole course of system running. A scheduling system can either construct its own information collection infrastructure, or employ existing information service systems, such as MDS and NWS. It is desired that the overhead introduced by the process of information collection is as small as possible.

2. Resource Selection

In principle, resource selection is performed in two steps. In the first step, the initial filtering is done with the goal of identifying a list of authorized resources that is available to a given application. Possibly, the initial list of authorized resources can be further refined by filtering according to the coarse-grained application requirements, such as hardware platform, operating system, minimum memory and disk space.

In the second step, those resources are aggregated into small collections such that each collection is expected to provide performance desired by the given application. The number of ways that the resources could be aggregated would be extremely large when the number of feasible resource is large. To overcome the complexity, different heuristics may be introduced.

3. Mapping

The third phase involves mapping the given set of applications onto a set of aggregated resources including both the computational resources and network resources. This is a well-know NP-complete problem and various heuristics may be used to reach a near-optimal solution. The effort of mapping in conjunction of resource selection produces a set of candidate schedules. Once the set of candidate schedules is ready, the scheduler starts to select the best schedule subject to the performance goal, based on mechanisms provided by the performance model.

4. Resource Allocation

Resource Allocation involves implementing the determined schedule. The job of resource allocation is performed by the resource allocation component introduced in Section 3.4.

3.5 Grid Scheduling Strategies

The scheduling problem for Grid environments is so complicated due to heterogeneous and dynamic nature of Grid resources besides the diversity of Grid applications. A variety of strategies have been developed to increase the effectiveness and efficiency of Grid scheduling.

Mapping heuristics

Given an application consisting of *n* jobs, either *independent* or *dependent*, and a set of heterogeneous resources, the problem of mapping the *n* jobs into the *m* resources subject to some criteria is a well-known NP-complete problem [63] . Due to the complexity, a large number of heuristics techniques have been proposed (refer to [63]).

As for an application consisting of *n* independent jobs, Braun et al. [38] conducted a comparison study on the mapping heuristics. The mapping heuristics discussed includes: Opportunistic Load Balancing, User-Directed Assignment, Fast Greedy, Min-min, Max-min, Greedy, Genetic Algorithm, Simulated Annealing, Genetic Simulated Annealing, Tabu, and A^{*}.

As for an application consisting of *n* dependent jobs specified by a directed acyclic graph (**DAG**), Kwok etc. [62] conducted a comprehensive survey on the problem of mapping **DAG**s to processors. Most scheduling heuristics are based on the so-called list scheduling technique. The basic idea of list scheduling is to make a scheduling list by assigning them some priorities, and then repeatedly execute the following two steps until all the nodes in the graph are scheduled: (a) remove the first node from the scheduling list; (b) allocate the node to a processor which allows the earliest start-time. The heuristics appearing in that survey are: Highest Level First, Longest Path, Longest Processing Time, Critical Path, and so on.

In general, these heuristics make different assumptions in order to make the scheduling effective. These assumptions, such as uniform job execution times, zero inter-job communication, contention-free communication, and full connectivity of parallel processors, does not hold in Grid environments. Thus when designing mapping algorithms based on these heuristics, careful modifications are necessary according to the characteristics of both Grid applications and Grid resources.

Resource reservation

The ability to determine if the desired set of resources is available at some time in the future is useful for scheduling. However, in general, a reservation-based strategy is limited due to the fact that currently deployed local resource management solutions do not support reservation. As a further step, Grid resources can be "reserved" in advance for a designated set of jobs. Such reservations operate much like a calendaring system used to reserve conference rooms for meetings. It must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached. This is done to meet deadlines and guarantee quality of service.

Rescheduling

After an application was scheduled, the performance of the application may not approach the desired performance due to the dynamic nature of the resources. It may be profitable to re-schedule the applications during execution to maintain good performance. At the minimum, an adequate Grid scheduler should acknowledge the resource failure and re-send lost work to a live computational resource. In summary, rescheduling is done to guarantee the job's completion and performance goal's achievement.

Rescheduling is an important issue for Grid schedulers, but no current system implements a complete set of rescheduling features. A mature scheduler with rescheduling should be able to adjust current schedules, move jobs from poorly performing nodes, and recover from failures.

Job Monitoring, Checkpointing and Migrating

To enable the important feature of rescheduling, three techniques, i.e., job monitoring, checkpointing and migrating, are highly important.

Job monitoring is responsible for detecting alert situations that could trigger a migration. This information is reported to the re-scheduler, which evaluates whether a migration is necessary, and in that case, decides a new allocation for the job.

Checkpointing is the capability of capturing periodically a snapshot of the state of a running job, which enables a job to be restarted from that state in a later time in case of migration. Checkpointing is beneficial for enabling us to resume a job instead of re-running it after its long execution. The main migration policies considered include performance slowdown, target system failure, job cancellation, detection of a better resource, etc.

3.6 Evaluation Criteria for Grid Scheduling Systems

In fact, it is quite difficult to make a comparison among different Grid scheduling systems, since each of them is suitable for different situations. For different Grid scheduling systems, the class of targeted applications and Grid resource configurations may differ significantly. In this subsection, a number of evaluation criteria for Grid scheduling systems are proposed.

- Application Performance Promotion

It involves reviewing how well the applications can benefit from the deployment of the scheduling system in the Grid environment.

- System Performance Promotion

The criterion of system performance promotion concerns how well the whole Grid system can benefit. For example, how much the utilization of resources is increased by, and how much the overall throughput gains.

- Scheduling Efficiency

It is desired that the Grid scheduling system can always produce good schedules. However, it is also required that the scheduling system should introduce additional overhead as low as possible. The overhead introduced by the scheduling system may exist in the information collection, the mapping process, and the resources allocation.

- Reliability

A reliable Grid scheduling system should provide some level of fault-tolerance. A Grid is a large collection of loosely-coupled resources, and therefore it is inevitable that some of the resources may fail due to diverse reasons. The scheduler should handle such frequent resource failures. For example, in case of resource failure, the scheduler should guarantee an application's completion.

- Scalability

Since a Grid environment is in nature heterogeneous and dynamic, a scalable scheduling infrastructure should maintain good performance with not only increasing number of applications, but also increasing number of participating resources with diverse heterogeneity.

- Applicability to applications and Grid environments

A scalable Grid scheduling system should be able to accommodate both a wider diversity of applications and a variety of Grid environments.

When designing the scheduling infrastructure of a Grid system, these criteria are expected to receive careful consideration. Emphasis may be laid on different concerns among these evaluation criteria according to practical needs in real situations.

4 Grid Scheduling Taxonomy

There have been a number of efforts attempting to design scheduling systems for Grid environments, each having its unique features. A comprehensive set of taxonomies is proposed in this section. The taxonomies are defined by many aspects of Grid scheduling systems. The set of taxonomies allows us to have a closer look at those features of interest related to Grid scheduling.

4.1 Knowledge of Application

- Application-level scheduling

The application-level scheduling scheme makes use of knowledge of applications as much as possible. Such kind of scheduling results in custom schedulers for each application attempting to maximize application performance, measured as runtime or speedup, with little regard to overall system performance. The complexity of application-level scheduling is the order of applications considered. AppLeS is a scheduling system that uses the application-level scheduling scheme [21].

- Resource-level scheduling

Resource-level scheduling does not use much knowledge of Grid applications. In this scheme, applications neither specify resource requirements nor provide application characteristics. Generally, a scavenging Grid aiming at leveraging the idle computing power will use this scheduling scheme. Condor is an example which uses resource-level scheduling [33].

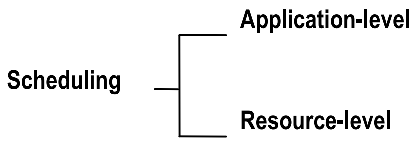


Figure 4-1: Knowledge of Applicaition

4.2 Inter-job Dependency

Given an application, the constituent jobs may either dependent or independent. The mapping algorithms for a set of independent jobs differ significantly from those for a set of dependent jobs. An application with a set of jobs is usually represented by a DAG. The mapping algorithms for a set of dependent jobs are more complicated.

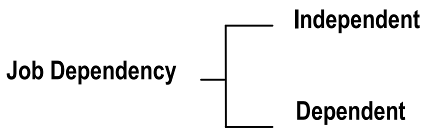


Figure 4-2: Job Dependency

Among existing Grid scheduling systems, some are intended to deal with application with independent jobs for simplifying the application model. Nimrod-G and Legion are two examples [21] [31] [32].

4.3 Information Service

The scheduler determines the state information of all the resources in a Grid system through the information service before making a scheduling decision. Different scheduling systems construct quite different structures to provide information service. Roughly, there are three categories as shown in Figure 4-3.

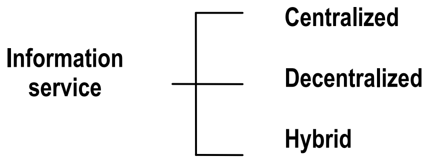


Figure 4-3: Inforamtion Service Organization

- **Centralized**

Under a centralized scheme, there exists a single centralized entity that maintains the state information of all resources. The centralized entity traverses every resource to get the most update state information periodically and keeps the information in its storage, waiting for queries issued by the schedulers. A centralized scheme is not scalable because it introduces the risk of single point of failure. Furthermore, a centralized entity may become the performance bottleneck when the entity can not afford a large number of resources and possible queries. Therefore, a centralized scheme is only suitable for small scale Grids. Globus MDS [24] is one example using centralized scheme.

- **Decentralized**

Within this scheme, every resource is responsible for maintaining its current state information locally, and answering queries from different clients. A decentralized scheme may not efficient due to the amplified quantity of queries. However the decentralized scheme is more reliable because the risk of single point of failure is removed through distributing the responsibility evenly to every resource. Although the decentralized scheme is suitable for large scale Grids, the large overhead should be carefully considered. NWS [28] uses the decentralized scheme.

- **Hybrid**

Using the hybrid scheme, resources are aggregated into several groups. Within each group, the centralized scheme is applied. Thus each group has one representative entity which is in charge of the information of all resources in its group. Over the groups, the decentralized scheme applies. This scheme is the most practical method for real wide-area Grid systems.

4.4 Online vs. Batch-mode

The scheduling can be performed in either online or batch mode. In the online mode, an application is scheduled onto resources as soon as it arrives at the scheduler. In the batch mode, an application is not scheduled onto resources when it arrives, and instead it will be hold in a queue that is examined for scheduling at prescheduled times called scheduling events. The online scheme considers an application for scheduling only once, whereas the batch-mode scheme considers an application for scheduling at each scheduling event until it is completed.

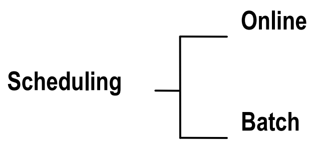


Figure 4-4: Scheduling Time

The scheduling overhead of the online scheme is much greater than that of the batch-mode scheme, because whenever the scheduling is performed, resource information should be retrieved and the scheduling procedure should be executed. So the online scheme is more suitable for systems with low arrival rates such that an application need not wait until the next scheduling event to begin its execution.

4.5 Scheduler Organization

The Grid scheduler organization can be classified into three categories: *centralized, decentralized,* and *hierarchical.*

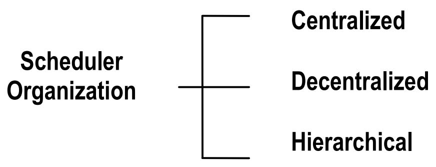


Figure 4-5: Scheduler Organization

- **Centralized**

In the centralized scheme, all user applications are sent to the centralized scheduler. There is a queue in the centralized scheduler for holding all the pending applications. When a user application is submitted to the scheduler, it may not be scheduled at once. Instead, it will be put in the queue, waiting for scheduling and resource allocation. Typically, each site neither maintains its queue nor performs any scheduling decision. A site receives jobs from the scheduler and executes them.

The centralized scheme is not very scalable with increasing number of resources. The central scheduler may prove to be a bottleneck in some situation. For example, if the scheduler is cut off due to a network failure, system availability and performance will be greatly affected. As an advantage, the scheduler is conceptually able to produce very efficient schedules, because the central scheduler has an overview on the available resources and on pending applications.

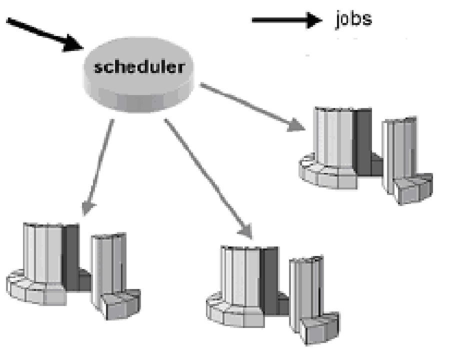


Figure 4-6: Centralized Scheduling [23]

- **Decentralized**

As depicted in Figure 4-7, a decentralized scheme distributes the responsibility of scheduling to every site. Each site in the Grid acts as both a scheduler and a computational resource. User applications are submitted to the local Grid scheduler where the applications originate. The local scheduler is responsible for scheduling its local applications, thus it possibly maintains a local queue to hold its own pending applications. Meanwhile, it should be able to respond to other schedulers' requests by acknowledging or denying it.

Since the responsibility of scheduling is distributed, the failure of a single scheduler does not affect others' working. So the decentralized scheme delivers better fault-tolerance and reliability than the centralized scheme. But the lack of a global scheduler, which knows the information of all applications and resources, usually results in low efficiency. Nevertheless, different scheduling policies on the local sites are possible. Therefore, site-autonomy can be achieved easily as the local schedulers can be specialized for the site owner's needs

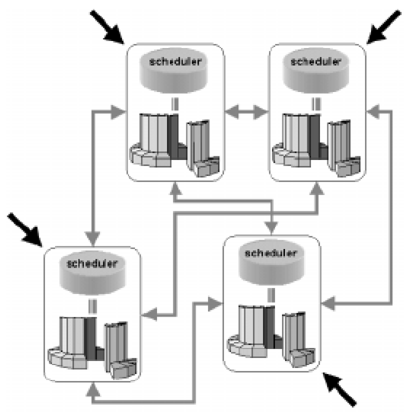


Figure 4-7 Decentralized Scheduling [23]

- **Hierarchical**

In the hierarchical scheme, the scheduling process is shared by different levels of schedulers. As shown in Figure 4-8, the higher-level schedulers manage larger sets of resources and lower-level schedulers manage smaller sets of resources. A higher-level scheduler has no direct control of a resource if there is one lower-level scheduler between the higher-level scheduler and the resource. A higher-level scheduler can only consider the capability of the set of resources managed by a lower-level scheduler as a whole entity, and utilizes the capability through invoking the lower-level scheduler.

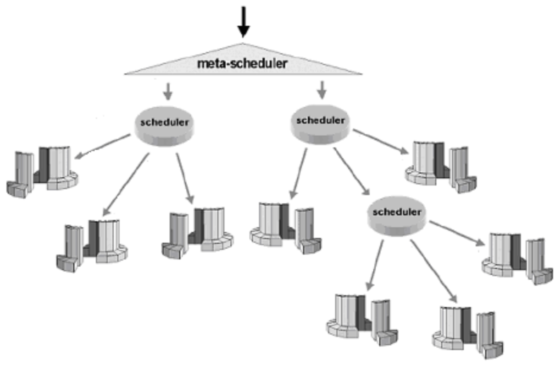


Figure 4-8: Hierarchical Scheduling [23]

Compared with the centralized scheduling, hierarchical scheduling addresses the scalability and the problem of single-point-of-failure. Nevertheless, it also retains some of the advantages of the centralized scheme. One of the issues with the hierarchical scheme is that it does not provide site autonomy yet. This might be a severe drawback in real Grid environments

4.6 Rescheduling

Rescheduling is an important technique used to enhance system reliability and flexibility. As show inFigure 4-9, Grid scheduling systems can be classified into two categories: one with re-scheduling support and the other without re-scheduling function.

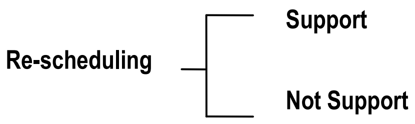


Figure 4-9: Re-scheduling Capability

4.7 Scheduling Policy

The scheduling policy determines how the scheduling should be performed. The performance goal defined in the scheduling policy plays a particularly important role in a Grid scheduling system. According to the different performance goals, the scheduling systems can be classified into three categories: *application centric, system-centric* and *economy-based,* as shown in Figure 4-10.

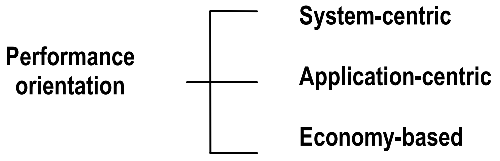


Figure 4-10: Performance Orientation

- **Application-centric**

Scheduling systems that fall in application-centric category try to favor the performance of individual applications. Typical performance goals sought by application-centric scheduling systems include minimizing execution time, maximizing the speed-up, etc. For example, an application-centric scheduling system will exploit a greedy mapping algorithm, which allocates the application to the resources that are likely to produce the best performance, without considering the rest of pending applications.

- **System-centric**

A system-centric scheduling system concerns the overall performance of the whole set of applications and the whole Grid system. The performance goals desired by a system-centric typically include resource utilization, system throughput, and average application response time. In a system-centric scheduling system, the process of ordering on the pending applications is usually performed in order to achieve a higher system-centric performance.

- **Economy-based**

An economy-based scheduling system introduces the idea of market economy. Under this scheme, scheduling decisions are made based on the economy model. The economy model defines: each application has the desired QoS, such as execution time and deadline, and the cost that the application will pay for the desired QoS; each resource is specified by its cost and the capacity. For each application, it wants to get as higher QoS as possible with the budget constraint. For each resource, it wants to get profit as much as possible by keep itself busy. Nimrod-G [20] is a scheduling system which exploits the idea of economy mechanism.

5 Existing Grid Scheduling Systems

To date, there have been a number of exciting initial efforts at developing scheduling systems for Grid environments. In this section, we focus on a representative group of these pioneering efforts to illustrate the state-of-the-art in Grid schedulers. It is often difficult to make comparisons between distinct efforts because each scheduler is usually developed for a particular system environment with different assumptions and constraints. In the following section, I attempt to outline the features of each system and summarize their advantages and drawbacks. Also how does they fit the taxonomy.

5.1 Information Collection Systems

The information service infrastructure plays a particularly important role in a scheduling system. *Meta Director Service* (**MDS**) from Globus and *Network Weather Service* (**NWS**) are two popular systems serving to provide the information publication and collection of resources in a grid system.

5.1.1 MDS (Meta Directory Service)

The Globus Metacomputing Directory Service (**MDS**) [24] provides is a LDAP-based information service infrastructure for computational Grids. It is used to collect and publish status information of Grid resources. Examples of the information that can be retrieved from an MDS server include operating system type, processor type and speed, number of processors, and available memory size of resources. .

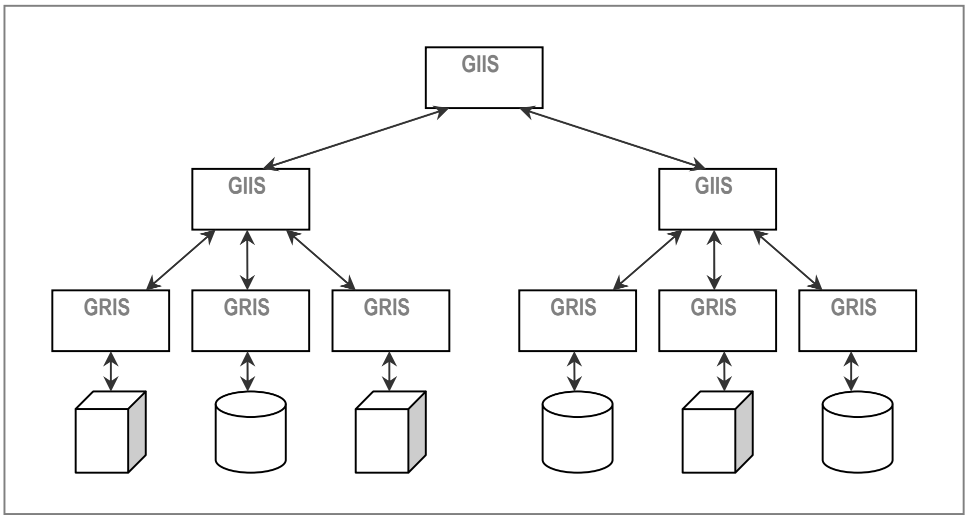


Figure 5-1: MDS Architecture

The MDS architecture is shown in Figure 5-1. Each GRIS (Grid Resource Information Service) is responsible for monitoring one resource and keeping all the current configuration, capabilities, and status of the resource. A GRIS can be queried for the information of a local resource. Each GIIS provides a means of aggregating information from several GRISs to present an information pool for a set of resources. Different GIIS can be further aggregated to manage more integrated information of a larger set of resources.

Thus, MDS provides a hierarchical method for providing information service in a Grid system, which is scalable and efficient. As a limitation, MDS can only provide current state information of the resources.

5.1.2 NWS (Network Weather Service)

The Network Weather Service (**NWS**) [28] is a distributed system that monitors and forecasts the performance of network and computation resources. Examples of the information that can typically be retrieved from an NWS server include the fraction of CPU available to a newly started process, the available memory size, and the bandwidth with which data can be sent to a remote host.

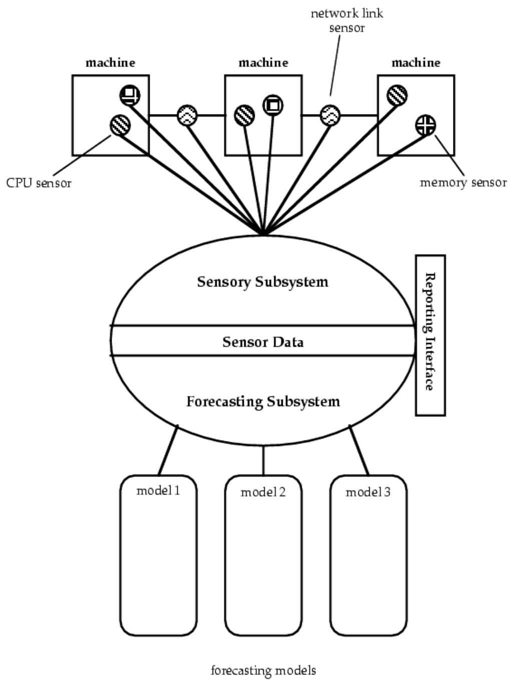


Figure 5-2: NWS Architecture[28]

Figure 5-2 depicts the architecture of Net Weather Service. Each resource has one NWS agent, which consists of three components: sensory subsystem, forecasting subsystem and reporting interface. NWS sensors collect the current performance that a resource is able to deliver at present. The collected performance data from sensors is input to the forecasting subsystem, and based on the forecasting models, different levels of performance in a future timeslot are forecast. Through the report interface, the predicted resource performance can be retrieved.

To avoid becoming a bottleneck of a system, the NWS does not have a centralized node that has the overall information of all the resources. The responsibility of information service is distributed to every resource. Superior to MDS, NWS provides not only the current performance information, but also the predicted performance information for a future timeslot which is more beneficial for estimating the performance behavior of a job running on the resource.

5.2 Conder

Condor [35] aims to increase utilization of workstation by hunting idle workstations for sharing job execution. Figure 5-3 shows the Condor scheduling structure.

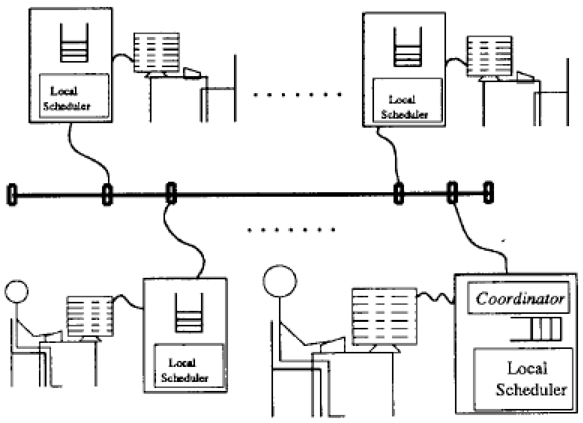


Figure 5-3: The condor Scheduling Structure[35]

Condor follows an approach to scheduling that lies between the centralized and decentralized scheme. For information collection, on one hand, Condor employs a *Coordinator* responsible for managing the set of available idle workstations. For scheduling jobs, on the other hand, each workstation itself is responsible for maintaining the local queue of jobs to be run and scheduling the jobs onto idle workstations for execution.

Condor is capable of checkpointing and job migrating, which are important features for rescheduling. In Condor, a remote job can run on a workstation only when the workstation is idle, that is, the workstation has no local workload. Once a workstation has its own workload, the remote job currently running on the workstation is preempted. With the help of checkpointing, the preempted job can be rescheduled to another idle workstation to resume its job, such that the previously accomplished results can be utilized.

The performance goal of Condor is to maximizing the throughput of the system, which is system-centric. The target applications of Condor are independent, non-realtime batch jobs. The underlying resources are homogeneous, preemptive, non-dedicated, and non-time-shared. The description of resources is coarse-grained since only the availability of workstations is considered.

Condor supports site autonomy. However, communication overhead of transferring a job is not considered. It is only suitable for WAN-based environment.

5.3 Condor-G

A newly proposed version of Condor, Condor-G [29], leverages the advantages of both condor and Globus Toolkit [4] [25]. Figure 5-4 shows the scheduling structure of Condor-G. Globus Toolkit is a software infrastructure for setting up a Grid environment across multiple administrative domains, which supports resources management, secure file transfer, information discovery and secure authentication and authorization in such a Grid environment. Based on Condor, Condor-G makes use of the mechanisms provided by Globus Toolkit to cross the boundaries of real institutions, aiming at utilizing the idle workstations among these institutions. Also the job creation, job monitoring and result collection are heavily relied on the **GRAM** (Grid Resource Access Management) component of Globus.

In condor-G, each *Job Submission Machine* constructs a GridManager locally which manages local jobs, retrieves the available resources, and schedules the jobs onto the feasible resources. **GSI** (Grid Security Infrastucture) mechanism of Globus is used by GridManager to do authentication and authorization with remote resources. Information collection of resources is based on **MDS** (Meta Directory Service) mechanism of Globus, which is in principle centralized.

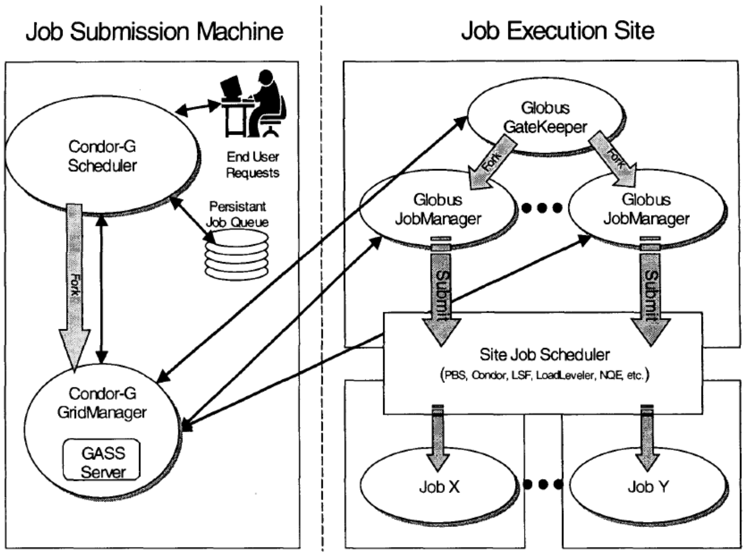


Figure 5-4: Condor-G scheduling Structure [29]

The problem of scheduling a set of dependent jobs is solved by Condor-G by designing the local GridManager with the coordinating function. Thus the applications running on Condor-G is more fine-grained compared to that of Condor.

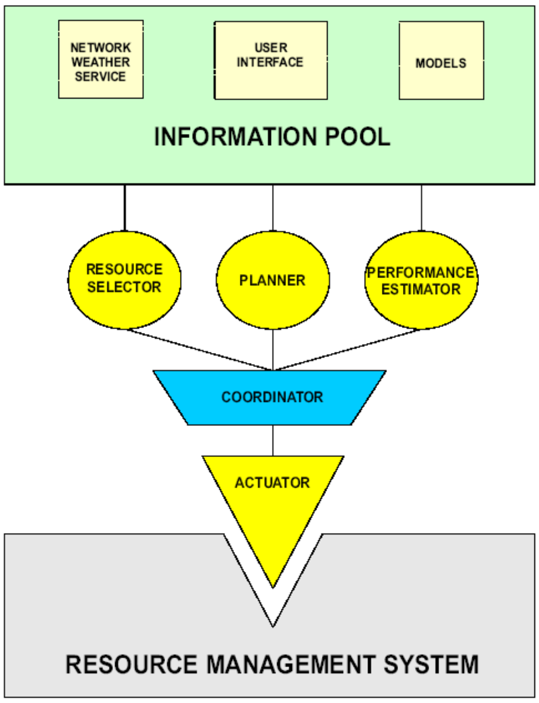
Resource heterogeneity is allowed in Condor through deploying the standard resource manager on resources. The computational resources in Condor could vary from workstations to clusters. But the description of resources remains coarse-grained.

Similar with Condor, Conder-G has the performance goal as maximizing the utilization of resources. Condor-G is resource-fault tolerant, meaning that it is able to cope with the resource failure. Condor-G allows inter-domain operation on remote resources that require authentication.

5.4 AppLeS

AppLeS [21] is an agent-based scheduling system for Grid environments, which targets to promote the performance of every individual application. ✌□□☹♏⧫ ♓⬧ ♌♋⬧♏♎ □■ ⧫♒♏ ♋□□●♓♍♋⧫♓□■⬄●♏❖♏● ⬧♍♒♏♎◆●♓■♑ □♋□♋♎♓♑❍ ♓■ ⬧♒♓♍♒ ♏❖♏□⍓⧫♒♓■♑ ♓■ ⧫♒♏ ⬧⍓⬧⧫♏❍ ♓⬧ ♏❖♋●◆♋⧫♏♎ ♓■ ⧫♏□❍⬧ □♐ ♓⧫⬧ ♓❍□♋♍⧫ □■ ⧫♒♏ ♋□□●♓♍♋⧫♓□■⬀ ☞□□ ♏♋♍♒ ♋□□●♓♍♋⧫♓□■⬂ its performance goal is specified by the application itself.

☜♋♍♒ ✌□□☹♏⧫ ♋□□●♓♍♋⧫♓□■ ♒♋⬧ ♓⧫⬧ □⬧■ ⬧♍♒♏♎◆●♏□ ⬧♒♓♍♒ ♎♏⧫♏□❍♓■♏⬧ ♋■♎ ❽♋♍⧫◆♋⧫♏⬧❾ ♋ ⬧♍♒♏♎◆●♏⬀ ❄♒♏ ⬧♍♒♏♎◆●♏ ♓⬧ ♍□❍□◆⧫♏♎ based □■ ⧫♒♏ ♋□□●♓♍♋⧫♓□■ ♍♒♋□♋♍⧫♏□♓⬧⧫♓♍⬧⬂ □♏□♐□□❍♋■♍♏ ❍□♎♏●⬂ ♋■♎ □♏⬧□◆□♍♏⬧ ♍◆□□♏■⧫●⍓ ♋❖♋♓●♋♌●♏ ⧫□ ⧫♒♏ ♋□□●♓♍♋⧫♓□■⬀


NETWORK WEATHER SERVICE
USER INTERFACE
MODELS
INFORMATION POOL
RESOURCE SELECTOR
PLANNER
PERFORMANCE ESTIMATOR
COORDINATOR
ACTUATOR
RESOURCE MANAGEMENT SYSTEM

☞♓♑◆❒♏ 🗄📫🗄🖳 ⚐❒♑♋■♓⌘♋⧫♓□■ □♐ ✌◻◻☹♏💧 💧♍♒♏♎◆●♏❒ ✌♑
♏■⧫ ☯🗏📂☸

✌⬧ shown ♓■ ☞♓♑◆❒♏ 🗄📫🗄📪 ♋■ ✌◻◻☹♏💧 ♋♑♏■⧫ ♓⬧ □❒♑♋■♓⌘♏♎ ♓■ ⧫♏❒❍⬧ □♐ ♐□◆❒ ⬧◆♌⬧⍓⬧⧫♏❍⬧ ♋■♎ ♋ ⬧♓■♑●♏ ♋♍⧫♓❖♏ ♍□❍◻□■♏■⧫ ♍♋●●♏♎ ⧫♒♏ 👍□□❒♎♓■♋⧫□❒📬 ❄♒♏ ♐□◆❒ ⬧◆♌⬧⍓⬧⧫♏❍⬧ ♋❒♏🖳

– ❄♒♏ ☼♏⬧□◆❒♍♏ 💧♏●♏♍⧫□❒📪 ⬥♒♓♍♒ ♍♒□□⬧♏⬧ ♋■♎ ♐♓●⧫♏❒⬧ ♎♓♐♐♏❒♏■⧫ ❒♏⬧□◆❒♍♏ ♍□❍♌♓■♋⧫♓□■⬧ ♐□❒ ⧫♒♏ ♋◻◻●♓♍♋⧫♓□■❼⬧ ♏⌧♏♍◆⧫♓□■📬
– ❄♒♏ 🏱●♋■■♏❒📪 ⬥♒♓♍♒ ♑♏■♏❒♋⧫♏⬧ ♋ ❒♏⬧□◆❒♍♏📫♎♏◻♏■♎♏■⧫ ⬧♍♒♏♎◆●♏ ♐□❒ ♋ ♑♓❖♏■ ❒♏⬧□◆❒♍♏ ♍□❍♌♓■♋⧫♓□■📬
– ❄♒♏ 🏱♏❒♐□❒❍♋■♍♏ ☜⬧⧫♓❍♋⧫□❒📪 ⬥♒♓♍♒ ♑♏■♏❒♋⧫♏⬧ ♋ ◻♏❒♐□❒❍♋■♍♏ ♏⬧⧫♓❍♋⧫♏ ♐□❒ ♍♋■♎♓♎♋⧫♏ ⬧♍♒♏♎◆●♏⬧ ♋♍♍□❒♎♓■♑ ⧫□ ⧫♒♏ ◆⬧♏❒❼⬧ ◻♏❒♐□❒❍♋■♍♏ ❍♏⧫❒♓♍📪 ♋■♎
– ❄♒♏ ✌♍⧫◆♋⧫□❒📪 ⬥♒♓♍♒ ♓❍◻●♏❍♏■⧫⬧ ⧫♒♏ ❻♌♏⬧⧫❼ ⬧♍♒♏♎◆●♏ □■ ⧫♒♏ ⧫♋❒♑♏⧫ ❒♏⬧□◆❒♍♏ ❍♋■♋♑♏❍♏■⧫ ⬧⍓⬧⧫♏❍⬧📬

The Network Weather Service (NWS) is used to provide dynamic information of resources and predict the resource load for the time frame in which the application will be scheduled. The User Interface provides specific information about the structure, characteristics and current implementations of the application and its jobs, as well as information on the user’s criteria for performance, execution constraints, and preferences. Finally, Models provide a repository of default application class models and application-specific models to be used for performance estimation.

❄♒♏ ⧫♋❒♑♏⧫ class □♐ ♋◻◻●♓♍♋⧫♓□■⬧ ♓■ ✌◻◻☹♏💧 ♒♋⬧ ♋ ♍□❍❍□■ ⬧⧫❒◆♍⧫◆❒♏🖳 ❍♋⬧⧫♏❒⁄⬧●♋❖♏📪 ❄♒♏ ♏⌧♏♍◆⧫♓□■ ⧫♓❍♏ ❍□♎♏● □♐ ⧫♒♏ ♍●♋⬧⬧ □♐ ♋◻◻●♓♍♋⧫♓□■⬧ ♍♋■ ♌♏ ♏⌧◻❒♏⬧⬧♏♎ ♋⬧ ♐□●●□⬧🖳

☞⌧♏♍❄♓❍♏ ◼ 💣♋⬧⧫♏❒☝□❍◻ ⌨ ❍♋⌧$_♓$ ✪ 💧●♋❖♏☝□❍◻$_♓$ “ ⌨ ☼ ♏⬧◆●⧫☝♋⧫♒♏❒

Where **MasterComp, SlaveCompi** and **ResultGather** provide a decomposition of the application execution behavior. AppLeS seems not to solve the multi-domain problem. Non-dedicated, time-shared resources are involved in AppLeS. In AppLeS, the performance goal is determined by the application itself. It is achieved through considering the application profile when selecting resources and making scheduling decision.

AppLeS employs NWS as its information service provider, which has a decentralized organization. Since each application has its own scheduler, it is obvious that the scheduler organization of AppLeS is evenly decentralized.

It is not difficult to note that there will be many AppLeS in a system simultaneously, each working on behalf on its own application. A worst-case scenario is that all of the AppLeS may identify the same resources as “best” for their applications and seek to use them simultaneously. Recognizing that the targeted resources are no longer optimal or available, they all might seek to reschedule their applications on another resource.

5.5 Legion

The Legion project [32] [33] from the University of Virginia is an object-based Grid environment, intended to connect a large suite of wide-area computational resources, with the illusion of providing a single virtual machine.

Legion is an object-oriented system consisting of independent disjoint objects that communicate with one another via method invocation. Classes define the types of their instances. An object is an instance of a class, which is responsible for managing a single resource. For example, HostClass encapsulates machine capabilities, e.g., CPU capability and memory size.

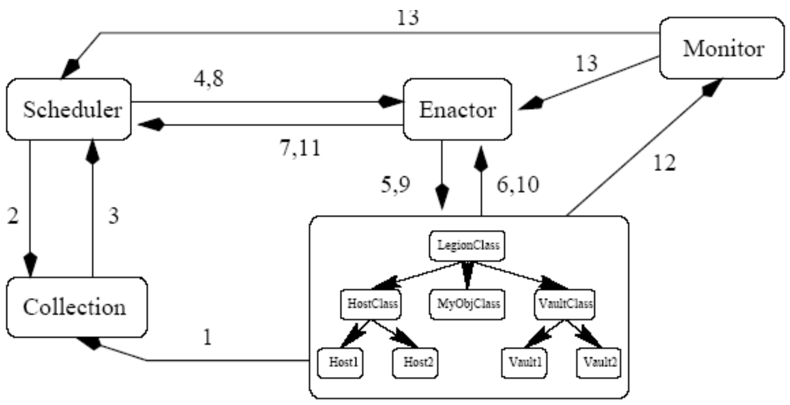


Figure5-6: Legion Scheduling Architecture [33]

In Legion, each application is typically scheduled by a customized scheduler associated to it. An application is basically also an object, and this object is to be instantiated into several instances. The scheduler is responsible for selecting a set of appropriate execution machines and mapping the set of instances onto the set of selected machines.

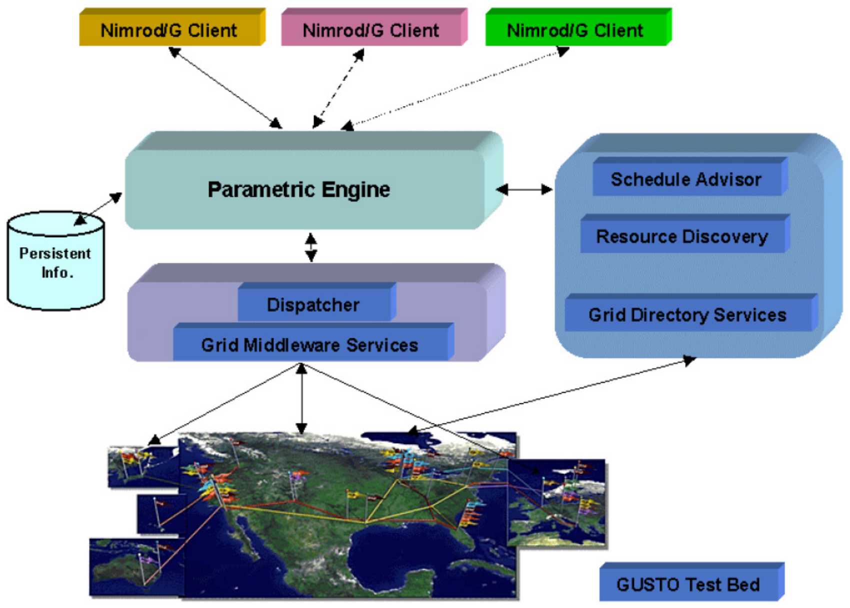
The Legion scheduling architecture is depicted in Figure 5-6. The Collection acts as a repository for information describing the state of the resources comprising the system. The Scheduler computes the mapping of instances in a class to resources. At a minimum, the Scheduler knows how many instances of each class must be started. The enactor involves implementing a schedule for a class forwarded from the scheduler.

Legion's targeted applications can be diverse since each application can develop its own scheduler. Due to the fact that the scheduler well knows the application-specific knowledge, the scheduler is able to produce efficient schedules for the application. The resources that Legion wants to utilize can be also heterogeneous. But it seems that Legion does not incorporate the dynamics in network behavior, which will lead to a drawback. Legion favors the performance goal of minimizing the execution time of an individual application.

Legion employs a centralized entity for information collection whilst it uses the decentralized scheme of scheduler organization. Through job monitoring, Legion is capable of rescheduling.

5.6 Nimrod-G

Nimrod [21] is a parametric study system, which uses a simple declarative parametric modeling language for expressing a parametric experiment. It has worked successful with a static set of computational resources, but is unsuitable as implemented in the large scale dynamic context of computational Grids, where resources are scattered across several domains.



☞♓♑◆❒♏ 🗄📬🖮💻 ❄♒♏ ✌❒♍♒♓♦♏♍♦◆❒♏ ❏♐ ☠♓❍❒❏♎📪☝ ☯📄📁☸

To overcome that shortcoming, the Nimrod/G [71] [20] makes use of the Globus toolkits for dynamic resource discovery and dispatches jobs over computational Grids. Nimrod/G supports an integrated computational economy in its scheduling system. This means that Nimrod/G can schedule applications on the basis of deadlines and budget.

In Nimrod/G, each application has one program and a large set of independent parameters to be studied, and hence it has a large number of independent jobs. An application specifies a deadline by which the application is expected to complete, and a price which the application owner will pay for the completion of the application. Each computational resource is specified a cost which the consumer should pay in order to use the resource. Briefly, a parametric study application is performed by Nimrod/G through the following steps:

1. **Discovery:** First the number and then the identity of the lowest-cost set of resources able to meet the deadline are identified. A cost matrix is used to identify low-cost resources; queries to the MDS directory service are then used to determine resource. The output from this phase is a set of resources to which jobs should be submitted, ordered by the cost to the application.
2. **Allocation:** Jobs are allocated to the candidate resources identified in Step 1.
3. **Monitoring:** The completion time of submitted jobs is monitored, hence establishing an execution rate for each resource.

4. **Refinement:** Rate information is used to update estimates of typical execution times on different resources and hence the expected completion time of the job. This refinement process may lead us to return to 1.

The scheme continues until the deadline is met, or the cost budget is exceeded. If the latter occurs, the user is advised and the deadline can be modified accordingly.

In Nimrod/G, the description of each application is coarse-grained. The targeted applications in Nimrod/G are specified with deadline and it is possible the deadline may not be met, and therefore they are soft realtime applications. The description of each resource is also coarse-grained. These resources are non-dedicated, timeshared and across multiple administrative domains.

Nimrod/G uses a hierarchical scheme of information service and a decentralized scheme of scheduler organization. Nimrod/G is useful for parametric study applications. Thus, the classes of applications supported are limited. It should be noted that in order to make the economy-based scheduling mechanism practically work, much work is still to be done.

5.7 Summary

The section is summarized in Table 5-1. This table is not intended to make a complete listing for each scheduling system, but extracts some features of interest for comparative purpose.

System	**Developer and year**	**Resources**	**Applications**	**Scheduling policy**	**Scheduling**
Condor	University of Wisconsin, Madison (1988)	Single-domain Grid, Non-dedicated, Non-time-shared	Single-job Application	System-centric, Maximizing Throughput	Centralized Information Service, Decentralized Scheduler Organization, Rescheduling-supported
Condor-G	University of Wisconsin, Madison (2001)	Multi-domain Grid, Non-dedicated, Non-time-shared	Single-job Application	System-centric, Maximizing Throughput	Centralized Information Service, Decentralized Scheduler Organization, Rescheduling-supported

AppLeS	University of California, San Diego (1996)	Single-domain Grid, Non-dedicated, Time-shared	Diverse Applications	Application-centric, Flexible	Decentralized Information Service, Decentralized Scheduler Organization, Rescheduling-supported
Legion	University of Virginia (1998)	Single-domain Grid Non-dedicated, Time-shared	Diverse ApPlications	Application-centric, Minimizing Execution Time	Centralized Information Service, Decentralized Scheduler Organization, Rescheduling-supported
Nimrod/G	Monash University, Australia (2000)	Multi-domain Grid, Non-dedicated, Time-shared	Soft Real-time, Parametric Study	Economy-based	Centralized Information Service, Decentralized Scheduler Organization, Non-rescheduling-supported

Table 5-1 Comparisons of Scheduling Systems

6 Conclusion and Future Work

6.1 Conclusion

As Grid computing becomes increasingly important, more and more efforts are focusing on the scheduling systems in Grid environments. The goal of this survey is to investigate issues and methods for designing effective and efficient scheduling systems for Grid environments.

The survey reviews the scheduling systems for cluster environments and shows that cluster scheduling systems cannot be applied to the Grid environments. On one hand, the Grid environments exhibit heterogeneous and dynamic characteristics, which cluster environments do not have. On the other hand, Grid environments are intended to execute much diverse applications compared to the cluster environments.

First, we investigate the challenges for designing scheduling systems in Grid environments, which are posed by the unique characteristics of Grid environments. The challenges include resource heterogeneity, site-autonomy, dynamic resource behavior, and application diversity. Next, a Grid schedule framework is proposed which consists of resource model, application model, performance model and scheduling policy. The framework is useful for guiding the design of a Grid scheduling system. Third, a common Grid scheduler architecture is presented. The architecture specifies different components each of which represents a necessary function for Grid scheduling. Next, the Grid scheduling procedure and strategies are discussed. Finally, some criteria for evaluating Grid scheduling systems are proposed.

A group of comprehensive taxonomies are defined by different aspects which outline the features of interest in Grid scheduling systems. At last, we survey a set of representative Grid scheduling systems. The survey shows that existing scheduling systems have many limitations. Each system may only work well under restricted system configurations, and apply to a limited body of applications.

6.2 Future Work

The research topic of Grid scheduling is quite young. No very successful scheduling systems has been proposed and reported, and a few proposed scheduling systems have many limitations. Therefore I believe that with the development and popularity of Grid computing, much work must be done in order to enable grid computing to be a real platform delivering high-performance services.

An idea has formed in my brain. Observe that all existing Grid scheduling systems have one common limitation: they all assume a flat organization, meaning that the scheduler has direct communication with each computational resource and therefore can allocate jobs to each resource directly. But it may not be the case in real environments. I believe that in the near future, many organizations may construct its local computational Grid to aggregate underutilized resources. Such a local Grid is based on high-speed local network, and therefore a centralized scheme should be successfully suitable for the local Grid. It is reasonably expected that parties outside the organization should not have direct access to the local resources; instead, they should utilized the local Grid through one entry point.

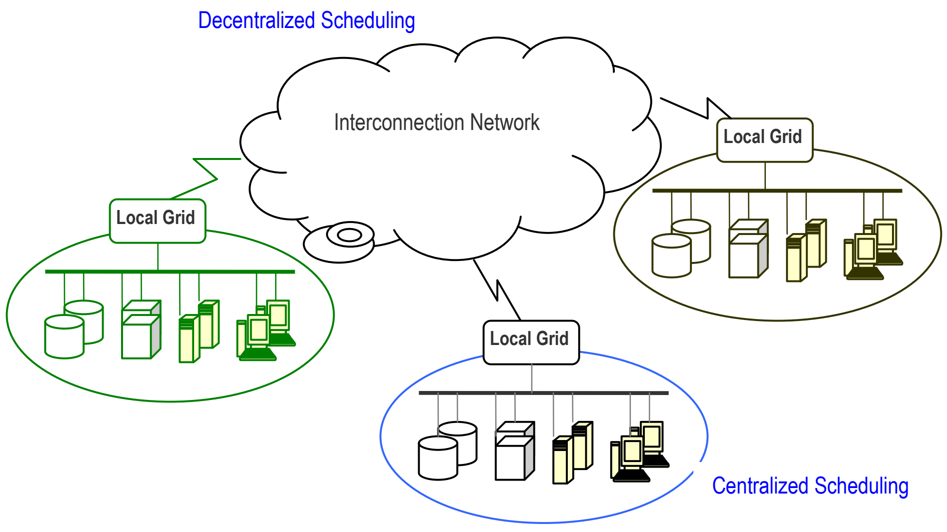


Figure 6-1: Two-level Grid Scheduling System

In such a context, **a two-level Grid scheduling system** will be very suitable. At the low level, a centralized scheme is used, while at the high-level, a decentralized scheme is used. Under this configuration, an application is submitted to the local Grid scheduler first. The local scheduler will then have two choices for handling the application: one is to schedule the application to the local resources, and the other one is to contact the other "local Grid scheduler" to determine whether it is beneficial to execute the application in a remote Grid.

Here are potential advantages of such a two-level Grid scheduling system.

1. Local resources are better protected since they are invisible outside the local Grid. It is only through a single point that the local Grid can be accessed.
2. Some local resources may be inaccessible in nature. For example, a machine with a private IP cannot be directly accessed by others. Therefore, such a resource cannot participate in a flat-organization Grid as is assumed by previously proposed scheduling systems. A two-level Grid scheduling mechanism overcomes this issue since others can utilize it though the local Grid scheduler which is globally visible.
3. Domain-autonomy is better realized. Each Local Grid can place its own scheduling policy.

Here are some research issues of such a two-level Grid scheduling system.

1. When application-centric performance goals such as execution time are concerned, how to predict the performance potential of an application on a remote Grid?
2. The factor of communication capability between two local Grids should be carefully incorporated when making a scheduling decision.
3. Whether co-allocation is possible? A co-allocation refers to allocating jobs of an application over several Grids simultaneously for execution.

In summary, such a two-level Grid scheduling system seems practical, beneficial, and flexible. Therefore it is worth conducting a research for it.

7 References

[1] I. Foster, C. Kesselman, and S. Tueche. *The anatomy of the Grid*. Intl. J. Super-computer Applications, 2001.

[2] B. Jacob, L. Ferreira, N. Bieberstein, C. Gilzean, J. Girard, R. Strachowski, S. Yu. *Enabling applications for Grid Computing with Globus*. Redbook, IBM Corp.

[3] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, O. Hernandez, J. Magowan, N. Bieberstein. *Introduction to Grid computing with Globbus*. Published by IBM as Redbook.

[4] I. Foster and C. Kesselman. *Globus: A metacomputing infrastructure toolkit.* International Journal of Supercomputer Applications, 11(2):115-128, 1997.

[5] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure.*

Morgan Kaufmann Publishers, 1998.

[6] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. *A security architecture for computational Grids. In ACM Conference on Computers and Security*, pages 83-91. ACM Press, 1998.

[7] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett. *From the I-WAY to the National Tech-nology Grid.* Communications of the ACM, 40(11):50-61, 1997.

[8] V. Berstis. *Fundamentals of Grid computing,* published by IBM as redbooks paper.

[9] F. Berman, G. Fox and A. Hey. *Grid computing: making the Global Infrastructure a reality.* WILEY, 2003.

[10] I. Foster, C. Kesselman, J. Nick and S. Tuecke. *The physiology of the Grid: an open Grid services architecture for distributed systems integration*. http://www.globus.org/research/ papers/ ogsa.pdf.

[11] L. Zhang, J. Chung and Q. Zhou. *Developing Grid computing applications, Part 1/Part2: Introduction of a Grid architecture and tookit for building Grid solutions.* http://www-106.ibm.com/developerworks/Grid/library/ws-Grid1/.

[12] James Frey. *Condor-G Tutorial*. http://www.bo.infn.it/calcolo/condor/condor-g.

[13] D. Roure, M. Baker, N. Jennings and N Shadbolt. The evolution of the Grid.

[14] F. Berman and R. Wolski. *Scheduling From the Perspective of the Application.* In proceedings of the Fifth IEEE Symposium on High performance Distributed Computing HPDC96, pages 100-111, Auguest 1996.

[15] F. Berman, R. Wolski, S. Figueira, J. Schopf and G. Shao. *Applicaion-level scheduling on distributed heterogeneous networks.* Super-computing 96.

[16] T. Casavant, and J. Kuhl. *A taxonomy of scheduling in general-purpose distributed computing systems.* IEEE transactions on Software Engineering 14, 2 (Feb 1998).

[17] *GGF workshop: Grid Scheduling Architecture.* http://www.Gridsched.org/.

[18] P. GradWell. *Overview of Grid Scheduling Systems.*

[19] H. G. Rotithor. Taxonomy of dynamic task scheduling schemes in distributed computing systems. IEE Proceedings on Computer and Digital Techniques, 141(1):1-10, Jan. 1994.

[20] Rajkumar Buyya, David Abramson, Jonathan Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*, The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000, Beijing, China. IEEE Computer Society Press, USA.

[21] David Abramson, Rok Sosic, J. Giddy, and B. Hall. *Nimrod: A tool for performing parameterised simulations using distributed workstations.* In HPDC, pages 112–121, 1995.

[21] F. Berman and R. Wolski. *The AppLeS Project: A Status Report*, 1997.

[22] Krauter, K., Buyya, R., Maheswaran, M., *A Taxonomy and Survey of Grid Resource management Systems.* Software-Practice and Experience, 32(2):135-164, 2002.

[23] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, Ramin Yahyapour: *Evaluation of Job-Scheduling Strategies for Grid Computing*. GRID 2000: 191-202.

[24] G. D. van Albada, J. Clinckemaillie, A. H. L. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. J. Overeinder, A. Reinefeld, and P. M. A. Sloot. *Dynamite - blasting obstacles to parallel cluster computing.* In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands,

number 1593 in Lecture Notes in Computer Science, pages 300{310, Berlin, April 1999. Springer-Verlag.

[25] Globus Project website, http://www.globus.org

[26] Global Grid Forum, http://www.ggf.org

[27] GGF's working group on Grid scheduling dictionary, http://www.fz-juelich.de/zam/RD/ coop/ggf/sd-wg.html.

[28] R. Wolski, N. T. Spring, and J. Hayes. *The Network Weather Service: A distributed resource performance forecasting service for metacomputing.* The Journal of Future Generation Computing Systems, Jan 1999.

[29] James Frey, Todd Tannenbaum, et al, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*. Journal of Cluster Computing, volume 5, pp. 237 -- 246, 2002.

[30] J. Gehring and A. Reinefeld. *MARS - a framework for minimizing the job execution time in a metacomputing environment*. Technical report, Paderborn Center for Parallel Computing, Jan 1995.

[31] G. Allen, D. Angulo, I. Foster, G. Lanfermann, and C. Liu. *The Cactus Worm: Experiments with dynamic resource discovery and allocation in a Grid environment*. International Journal of High Performance Computing Applications, 15(4), Jan 2001.

[32] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. *Resource management in legion. In 5th Workshop on Job Scheduling Strategies for Parallel Processing*, in conjunction with the International Parallel and Distributed Processing Symposium, Apr 1999.

[33] M. J. Lewis and A. Grimshaw. The core legion object model. In Proceedings of The Fifth IEEE International Symposium on High Performance Distributed Computing. IEEE Computer Society Press, August 1996.

[34] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. *Application level scheduling on distributed heterogeneous networks*. In Proceedings of Supercomputing 1996, Jan 1996.

[35] Michael Litzkow, Miron Livny, and Matt Mutka, *Condor-A Hunter of Idle Workstations*. In Proc. The 8th International Conference of Distributed Computing Systems, San Jose, California, June, 1988, pp.204-111.

[36] M. A. Baker, G. C. Fox, and H. W. Yau, *Cluster Computing Review*, Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.

[37] R. Buyya (ed.), *High Performance Cluster Computing: Systems and Architectures*, Prentice Hall, 1999.

[38] T.D. Braun, H.J. Siegel, N. Beck, L.B. ol. oni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys and B. Yao, *A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-machine Heterogeneous Computing Systems*, IEEE symposium on Reliable Distributed Systems, 1998, pp. 330-335.

[39] I. Ekmeci'c, I. Tartalja, and V. Milutinovi'c, *A survey of heterogeneous computing: Concepts and systems,* Proceedings of the IEEE, Vol. 84, No. 8, Aug. 1996, pp. 1127--1144.

[40] Lindahl, G., Grimshaw, A., Ferrari, A., and Holcomb, K. (1995). *Metacomputing --- What's in it for me?* http://legion.virginia.edu/papers/why.pdf.

[41] W. Hoschek, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger, Data *Management in an International Data Grid Project*, Proceedings of the first IEEE/ACM International Workshop on Grid Computing, (Springer Verlag Press, Germany), India, 2000.

[42] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F. Freund. *Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems.* Journal of Parallel and Distributed Computing, 59(2):107--131, November 1999.

[43] Rajkumar Buyya, Steve Chapin, and David DiNucci. *Architectural Models for Resource Management in the Grid*. In 1st IEEE/ACM Int. Workshop on Grid Computing (GRID 2000.

[44] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, 1998.

[45] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. *A directory service for configuring high-performance distributed computations*. In Sixth IEEE Syrup. on High Performance Distributed Computing, pages 365-375, 1997.

[46] I. Foster, J. Geisler, B. Nickless, and S. Tuecke. *Software infrastructure for the I-WAY metacomputing experiment.* Concurrency: Pract. Exper., 10(7):567--581, 1998.

[47] D. Abramson, R. Sosic, J. Giddy, and B. Hall. *Nimrod: A tool for performing parameterised simulations using distributed workstations.* In Proc. 4th IEEE Symp. on High Performance Distributed Computing. IEEE Computer Society Press, 1995.

[48] R. Ramamoorthi, A. Rifkin, B. Dimitrov, and K.M. Chandy. *A general resource reservation framework for scientific computing*. In Scientific Computing in Object-Oriented Parallel Environments, pages 283-290. Springer-Verlag, 1997.

[49] M.J. Gonzalez, Jr., *Deterministic processor scheduling*. ACM comput. Surv. 9(3), pp. 173-204, Sep 1977.

[50] Gregory Chun, Michael O. McCracken, Josh Wills, *Scheduling in Metacomputing Environments.*

[51] Hongtu Chen, Muthucumaru Maheswaran, *Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems.* IPDPS 2002.

[52] R. Buyya, D. Abramson, and J. Giddy. *An economy driven resource management architecturefor global computational power Grids*. In 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, Jan 2000.

[53] E. K. and DanWerthimer, D. Anderson, J. Cobb, and M. Lebofsky. *SETI@home: Massively distributed computing for SETI.* Scientific Programming, 2000.

[54] Raman, R., Livny, M., and Solomon, M. *Matchmaking: Distributed resource management for high throughput computing*. In *Proceedings of the 7th IEEE Symposium on High-Performance Distributed Computing* (July 1998).

[55] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. *Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms*. In Proc. of 8th IEEE Int. Symp. on High Performance Distributed Computing, pages 97–104, 1999.

[56] D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini and G. R. Nudd, *Local Grid Scheduling Techniques using Performance Prediction*.

[57] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*. In Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), pages 349--363, May 18, 2000.

[58] Holly Dail, Henri Casanova and Francine Berman, *A Modular Scheduling Approach for Grid Application Development Environments*, Submitted to Journal of Parallel and Distributed Computing, 2002.

[59] Ajith Abraham and Rajkumar Buyya and Baikunth Nath, *Nature's Heuristics for Scheduling Jobs on Computational Grids.*

[60] Ian Foster, *Computational Grids*.

[61] Peter Gradwell, *Overview of Grid scheduling Systems*.

[62] Y.-K. Kwok and I. Ahmad. *Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors*. ACM Computing Surveys, Vol. 31, No. 4, pp. 406--471, 1999.

[63] Gonzalez, M. J. *Deterministic Processor Scheduling*. ACM Computing Surveys, Vol. 9, Number 3, pp. 173-204, September 1997.

[64] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. *A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems*. In Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99), pages 15--29, Apr. 1999.

[65] CHAPIN, S. *Distributed and multiprocessor scheduling,*. ACMComputer Surveys 28, 1 (March 1996).

[66] Angulo, D. and Foster, I. and Liu, C. and Yang, L. *Design and Evaluation of a Resource Selection Framework for Grid Applications*. In http://www.globus.org/research/papers.html, 2002.

[67] K. Kennedy, M.Mazina, J. M. Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt, D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski, *Toward a Framework for Preparing and Executing Adaptive Grid Programs,* IPDPS 2002.

[68] Fracine Berman, *High-Performance Schedulers*, in *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, 1998.

[69] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine|A User's Guide and Tutorial for Network Parallel Computing*, MIT Press, 1994.

[70] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1995.

[71] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Glocal Grid? In Proceedings of IPDPS2000.

[72] D. Wright. *Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with condor.* In Proceedings of the Linux Clusters: The HPC Revolution conference, June 2001.

[73] M. Flynn, *Very high-speed computing systems*, Proceedings of the IEEE, 54 (1966) 1901–1909.