International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-2, Issue-1,January 2016
ISSN: 2395-3470
www.ijseas.com

Study on Resource Management and Scheduling in Computational Grid

V.M.Sivagami , K.S.Easwarakumar

[1]AssociateProfessor,Department of Information technology Sri Venkateswara College of Engineering
[2]Professor Department of Computer Science Engineering Anna University

ABSTARCT
Scheduling and Resource Management is an important issue in Grid Computing. There are many Scheduling algorithms existing in the literature. This paper discusses about a detailed study on different scheduling algorithms in Computational Grid. Grid scheduling could benefit from several traditional scheduling Algorithms. These Algorithms have achieved successful results in a wide range of scheduling applications. In a scheduling problem, the goal is to appropriately assign all the tasks that are requesting service to the available processors so that the time constraints are satisfied. The time constraints of a task are the task's deadline (which is the time by which it is desirable for it to complete execution) and the task's earliest starting time on each processor. Success ratio is the common measure for evaluating the scheduling performance which is , defined as the ratio of the number of tasks that are feasibly scheduled (that is, the tasks whose time constraints are met) over the total number of tasks requesting service. Effective computation and job scheduling is rapidly becoming one of the main challenges in grid computing and is seen as being vital for its success.

Keywords: Resource Management, Resource Discovery, Scheduling Algorithms.

INTRODUCTION

Efficient use of Grids has become an important problem. A Grid is a structure with distributed heterogeneous resources that are offered to users. Users submit jobs that should be efficiently processed using resources available on the Grid .The resources (e.g., machines, CPUs, memory, storage space) often have a limited capacity and their characteristics change in time (due for example to machine breakdown or consumption of storage space). The goal is to find an optimal placement of tasks with respect to the costs of the resources assigned. The cost function is often minimized, for example, a maximal estimated completion time of all tasks in the scheduling problem. The grid scheduler must make best effort decisions and then submit the jobs to the hosts selected, generally as a user. Furthermore, the grid scheduler does not have control over the set of jobs submitted to the grid, or local jobs submitted to the computing hosts directly. This lack of ownership and control is the source of many of the problems yet to be solved in this area. The grid scheduling is a particular case of tasks scheduling on machines problem. In the grid scheduling every machine can execute any task, but for different time. To make information available to users quickly and reliably, an effective and efficient resource scheduling mechanism is crucial. Generally grid resources are potentially very large in number with various individual resources that are not centrally controlled. These resources can enter as well as leave the grid systems at any time. For these reasons resource scheduling in large-scale grids can be very challenging.

The scheduler is responsible for selecting resources and scheduling jobs in such a way that throughput and cost of the resources utilized. It allows the user to specify the required resources and environment for job execution. A Grid

scheduler decides to which local resource manger(s) the job should be submitted[1,4].

General definition and terminology

Grid Scheduler: definition of a Grid scheduler will much depend on the way the scheduler is organized (whether it is a super-scheduler, meta-scheduler, decentralized scheduler or a local scheduler) and the characteristics of the environment such as dynamics of the system. In a general setting, however, a Grid scheduler will be permanently running as follows: receive new incoming jobs, check for available resources, select the appropriate resources according to feasibility (job requirements to resources) and performance criteria and produce a planning of jobs (making decision about job ordering and priorities) to selected resources. Usually the following terminology is employed for scheduling in Grids as cited in [7,8,12,13,34].

Scheduling Problem: A scheduling problem is specified by a set of machines, a set of tasks, an optimality criterion, environmental specifications, and by other constraints. The environment defines relations and connections between the machines and other used structures. A goal of the scheduling problem is to find an optimal schedule in the environment and to satisfy all constraints. [8].

Schedulable unit: Tasks could be independent (or loosely coupled) among them or there could have dependencies, as it is the case of Grid workflows. Job: A job is a computational activity made up of several tasks that could require different processing capabilities and could have different resource requirements (CPU, number of nodes, memory, software libraries, etc.) and constraints, usually expressed within job description. In the simplest case, a job could have just one task.

A machine (computing unit) is a set of cumulative resources (CPUs, memory, storage space, list of specializations) with limited capacities. A machine is described by a name, a set of resources, a list of specializations (e.g., architecture, accelerator), and by its capacity, load, speed and location. All these characteristics are called descriptors of the machine. [8]

A job (task, activity) is a basic entity which is scheduled over the resources. A job has specific requirements on the amounts and types of resources (including machines), or required time intervals on these resources, where the job can be scheduled.

Task: represents a computational unit (typically a program and possibly associated data) to run on a Grid node. Although in the literature there is no unique definition of task concept, usually a task is considered as an indivisible

The job j has the following variables (properties):

- **The processing** time is estimated as the due date of a chosen time queue (often too big), or by using statistics based on recent runs of the chosen program (e.g., Gaussian1, Amber2). Also the processing time is estimated according to the length of the input data or by other advanced techniques (e.g., fuzzy estimate) and it is denoted as Pj.

- **A release date (rj)** is the time when a job is available to start processing including i/o operations. The release date could be explicitly set (a static plan, an allocation in future) or unknown.

- **A due date (dj)** is implicitly set by the chosen time queue or explicitly by the user. Both cases are hard due dates.

- A **weight (wj)** signifies the importance of a job j, e.g., it may be set according to the user's group (project). It should reflect natural preferences, e.g., a local job has a greater weight than a global one.

• **A setup time (sj)** may be used to designate the time required for retrieving (copying) input data or the time for linking to a needed library. This may be dependent on the sequence of jobs.

• The **start time (Sj)** is a time when the job actually begins its processing . The **completion time (Cj)** is the time when a job completes its processing if preemptions are not allowed).

• **Planning:** A planning is the mapping of tasks, jobs and applications to computational resources.

Application: An application is a software for solving a (large) problem in a computational infrastructure; it may require splitting the computation into many jobs or it could be a "monolithic" application. In the later case, the whole application is allocated in a computational node and is usually referred to as application deployment. As in the case of jobs, applications could have different resource requirements (CPU, number of nodes, memory, software libraries, etc.) and constraints, usually expressed within application description.

Resource: A resource is a basic computational entity (computational device or service) where tasks, jobs and applications are scheduled, allocated and processed accordingly. Resources have their own characteristics such as CPU characteristics, memory, software, etc. Several parameters are usually associated with a resource, among them, the processing speed and workload, which change over time. As in the case of jobs and applications, resource characteristics are usually given by the resource description. It should be noted that in a Grid computing environment resources are geographically distributed and may belong to different administrative domains implying different usage policies and access rights.

Specifications: Task, job and application requirements are usually specified using high level specification languages (meta-languages). Similarly, the resource characteristics are expressed using specification languages. One such language is the ClassAds language [56].

Resource pre-reservation: The pre-reservation is needed either when tasks, jobs or applications have requirements on the finishing time or when there are dependencies/precedence constraints that require advance resource reservation to assure the correct execution of the workflow. The advance reservation goes through negotiation and agreement protocols between resource providers and consumers provides details of the processing characteristics and the constraints. The field contains the objective to be minimized.

Machine Environment ( )

The basic notations of machine environments are single machine , identical machines in parallel (Pm), machines in parallel with different speeds (Qm), unrelated machines in parallel (Rm), and (flow|open|job) shop(Fm,Om, Jm).[8]

When 1 is in the field, it indicates that we are working with only one machine where jobs can be scheduled. A single machine can be seen as a disjunctive resource and other environments as cumulative resources. We could schedule the jobs on m identical machines in a parallel environment be unavailable due to maintenance or a breakdown. There are three different cases of working with unavailable machines

Resumable: The machine is resumable if the unfinished job can continue after the end of the unavailable interval without any penalty.

Non-resumable: The machine is non-resumable if the job must be fully restarted after the unavailable interval.

Semi-resumable: The machine is semi-resumable if the job must be partially restarted after the unavailable interval.

Online Scheduling : Online scheduling supposes that jobs arrive over time or one by one. Jobs are

scheduled without any knowledge of the future, and often without knowledge of the processing

Objective Function ()

Here we consider the objective functions that evaluate the quality of solutions. There are two groups of scheduling objective functions.

In the first group there are basic functions.

• **A makespan** defined as max (C1, . . . ,Cn), the completion time of the last job. This objective function formalizes the viewpoint of the owner of the machines. If the makespan is small, the utilization of the machines is high.

• **A total weighted completion time** or the total unweighted completion time defined as PCj This objective is more suitable for the user, the time it takes to finish individual jobs may be more important.

• **A maximum lateness** defined as max(L1, . . . ,Ln), where Lj = Cj − dj (the difference between the completion time and the due date).

A Traditional Scheduling View of Grid Scheduling

In traditional scheduling, scheduling is defined as the allocation of operations to resources over time, taking into account some performance measure criteria subject to the satisfaction of constraints [Pinedo, 1995]. So, is Grid scheduling a new problem which differs from the traditional scheduling? There is no clear evidence that it differs in any fundamental way from the traditional scheduling problems. The fundamental difference may be the dynamic nature of resources and constraints in the Grid environment, but this point is not clear at present.

Grid scheduling: Grid scheduling is the mapping of individual tasks to computer resources, while respecting service level agreements (SLAs), etc.

Constraints: scheduling constraints can be hard or soft. Hard constraints are rigidly enforced. Soft constraints are those that are desirable but not absolutely essential. Soft constraints are usually combined into an objective function. So, what are the hard and soft constraints in Grid scheduling?

Optimization Criteria: a variety of optimization criteria are of interest for Grid scheduling: minimization of the maximum lateness, minimization of the cost to he user ,maximization of the profit, maximization of personal or general utility, maximization of resource utilization, fairness, minimization of variance, maximization of robustness and predictability, minimization of broken SLAs, etc.

Scheduling data: a variety of data are necessary for a scheduler to describe the jobs and the resources: job length, resource requirements estimates, time profiles, uncertain estimate, etc.

Methodologies*:* the meeting agreed that there is no clear choice of method which will be the best for Grid scheduling.

Different Types of Schedulers

- **Grid Scheduler:** Software components in charge of computing a mapping of tasks, jobs or applications to Grid resources under multiple criteria and Grid environment configurations. Different levels within a Grid scheduler have been identified in the Grid computing literature comprising: super schedulers, meta-scheduler, local/cluster scheduler and enterprise scheduler. As a main component of any Grid system, Grid scheduler interacts with other components of the Grid system: Grid information system, local resource management systems and network management systems. It should be noted that in Grid environments, all these kinds of schedulers must coexists, and they could in general pursue conflicting goals,

thus, there is need for interaction between the different schedulers in order to execute the tasks as cited in [13,34].

- **Super-scheduler:** This kind of schedulers corresponds to a centralized scheduling approach in which local schedulers are used to reserve and allocate resources in the Grid. The local schedulers manage their job queue processing. The super-scheduler is in charge of managing the advance reservation, negotiation and service level agreement. Notice that tasks, jobs or applications are entirely completed in unique resource.
- **Meta-scheduler:** This kind of schedulers (also known as Meta-broker in the literature) arise when a single job or application is allocated in more than one resource across different systems. As in the case of super-schedulers, a meta-scheduler uses local schedulers of the particular systems. Thus, meta schedulers coordinate local schedulers to compute an overall schedule. Performing load balancing across multiple systems is a main objective of such schedulers.
- **Local/Cluster Scheduler:** This kind of scheduler is in charge of assigning tasks, jobs or applications to resources in the same local area network. The scheduler manages the local resources and the local job queuing system and is this a “close to resource” scheduler type.
- **Enterprise Scheduler**: This type of scheduler arises in large enterprises having computational resources distributed in many enterprise departments. The enterprise scheduler uses the different local schedulers belonging to the same enterprise.
- **High-throughput schedulers:** The objective of this kind of scheduler is to maximize the throughput (average number of tasks or jobs processed per unit of time) in the system. These schedulers are thus task-oriented schedulers, that is, the focus is in task performance criteria.
- **Resource-oriented schedulers:** The objective of this kind of scheduler is to maximize resource utilization. These schedulers are thus resource-oriented schedulers, that is, the focus is in resource performance criteria.
- **Application-oriented schedulers:** This kind of schedulers are concerned with scheduling applications in order to meet user’s performance criteria. To this end, the scheduler have to take into account the application specific as well as system information to achieve the best performance of the application. The interaction with the user could also be considered.

User-centric scheduler:

AppLeS is a User-centric Scheduler.

- The User-centric Performance goals can be defined as follows:
- Minimize execution time
- Minimize the waiting time in the ready-to-process queue
- Maximize speed-up on the user’s own platform i.e. minimize the turn-around time
- Minimize process slow-down, defined as the ratio of turn-around time to the actual execution time.

Service-provider centric scheduler**:** A service-provider centric scheduler does not try to obtain detailed information about the characteristics of the application and it tries to optimally use the computing power of the grid. Condor [2] is an example of a service-provider centric scheduler. The Service-centric Performance goals can be defined as follows:

- Maximize throughput. (Throughput is the number of jobs processed per unit of time.)
- Maximize utilization. (Utilization is the percentage of time a resource is busy)
- Minimize flow time. (Flow time or session time is the sum of completion time of all jobs.)

Economy based scheduler: A third case of scheduling is called Economy-based. The scheduler mimics the market place where the user's application is to execute at a particular total maximum cost or it is required to meet certain deadlines. The user is interested in satisfying the requirements of the application at the minimum possible cost. The goal of the service-provider is to obtain the highest price for its services. Thus each resource is characterized by its cost and its computational capacity characteristics. The scheduler tries to provide the service to the user at the lowest cost while providing maximum returns to the service provider. Nimrod-G is a scheduling system based on such market-like considerations.

General Classification of Scheduling

1. Centralized Scheduling
2. Decentralized Scheduling
3. Hierarchical Scheduling

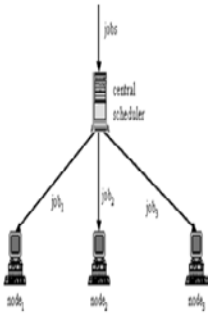
Centralized Scheduling

In a centralized scheduling scheme, a central machine acts as a resource manager which is responsible to schedule jobs to all the nearby nodes which belongs to the group. This scheme is often used in situations like a computing center where resources have similar characteristics and usage policies. [1]

Benefits and Limitations of a Centralized Scheduling System

- The scheduler may produce better scheduling decisions because it has all necessary, and up-to-date, information about the available resources.
- But, it does not scale well with the increasing size of the environment that it manages. The scheduler itself may well become a bottleneck, and it presents a single point of failure in the environment. Centralized scheduling structures are difficult to maintain and reconfigure, inefficient to satisfy real-world needs, and costly in the presence of failures. Also, the amount of knowledge to manage is very large.[1]

Decentralized Scheduling



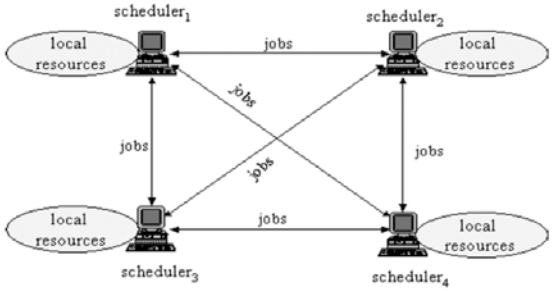
In Decentralized scheme, scheduling of jobs is done in a distributed manner, and the jobs are distributed to multiple localized schedulers, which interact with each other in order to dispatch jobs to the participating nodes. The scheduler communicates with other schedulers in two different ways either through direct or indirect communication.

Benefits and Limitations of a Distributed Scheduling System

- It overcomes scalability problems
- It can offer better fault tolerance and reliability.
- However, the lack of a global scheduler, which has all the necessary information on available resource, usually leads to sub-optimal scheduling decisions.[1]

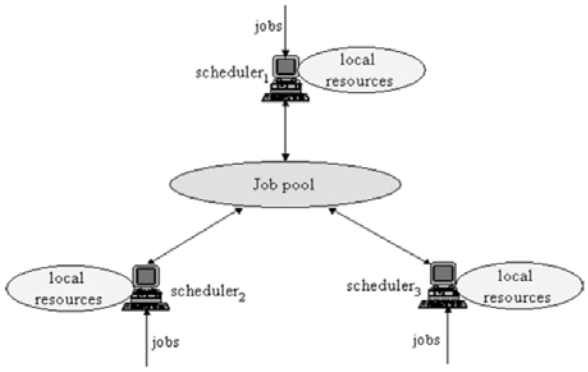
Distributed Scheduling-Direct communication

In Direct Communication, each local scheduler directly communicates with other schedulers for job dispatching. Each scheduler has a list of remote schedulers that they can interact with, or there may exist a central directory that maintains all the information related to each scheduler. If a job cannot be dispatched to its local resources, its scheduler will communicate with other remote schedulers to find resources.[1]



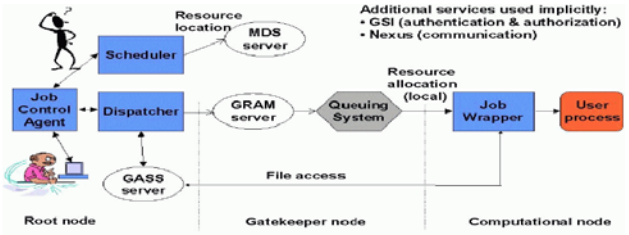
Communication via a central job pool

In this scheme, jobs that cannot be executed immediately are sent to a central job pool. Compared with direct communication, the local schedulers can potentially choose suitable jobs to schedule on their resources. Policies are required so that all the jobs in the pool are executed at some time.

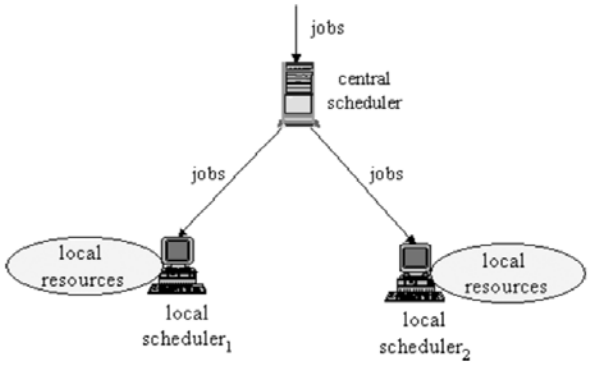


Hierarchical scheduling

In hierarchical scheduling, a centralized scheduler interacts with local schedulers for job submission. The centralized scheduler is a kind of a meta-scheduler that dispatches submitted jobs to local schedulers. The disadvantage of hierarchical scheduling is it has scalability and communication problem similar to centralized



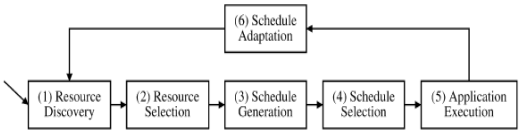
scheduling. But, compared with centralized scheduling, one benefit in hierarchical scheduling is that the global scheduler and local scheduler can have different policies in scheduling jobs.[1]



Existing Schedulers

AppLeS

The AppLeS project was based at UC San Diego and was focused on development of scheduling agents for applications running in Grids. The AppLeS system collects resource information from the Network Weather Service (NWS) running at each computing node, and dispatches tasks to lighter loaded nodes; while scheduling actual execution of applications is local. AppLeS uses other RMSs, e.g. Globus, Legion, or NetSolve, to execute actual jobs (i.e. it can be viewed as a meta-middleware placed above the standard Grid middleware). Each application has embedded AppLeS agents that perform resource scheduling. [14,18]



Nimrod/G

Nimrod/G is a Grid resource broker based on an economy-driven approach to manage resources and schedule jobs. It utilizes services provided by other Grid middleware (e.g. Globus, Legion, Condor), and the GRACE trading mechanisms. Note that, while at the Nimrod/G site there are references to work completed in 2007, the latest version of this middleware v3.0.1 was released in October, 2005. Thus, to the best of our knowledge, today this project is no longer active.

OpenPBS

OpenPBS is a simple workload management solution intended for small clusters of dedicated homogeneous nodes. Here, computers are federated into a virtual pool of resources. Workload is scheduled to run within this virtual pool, based on simple scheduling algorithms. OpenPBS is one of workload managers accessible from the CSF meta-scheduler. However, the last release of the OpenPBS as an independent project happened in 2001. At the same time, the PBS Professional, is the commercial product developed and sold by the Altair corporation.

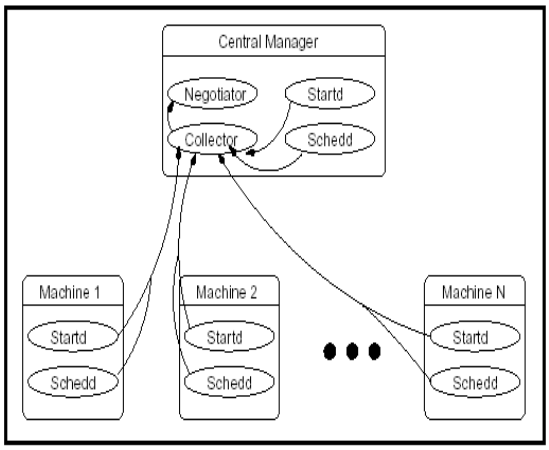
NetSolve

The NetSolve project was focused on execution of scientific applications in heterogeneous environments, while utilizing different scheduling algorithms for different applications. Job completion time estimation was based on performance and load models, while a dynamic job queue was used for job ordering. Length of this queue was adaptively adjusted based on historical performance data (an example of a system-level scheduling. In addition, mechanisms for scheduling multi-step, data-dependent jobs have been implemented. Recently, the NetSolve project has been extended to Grids through the GridSolve infrastructure. Both projects are active; for instance, a new release of GridSolve software appeared on 2008.

Condor

Condor is a high-throughput computing environment that manages large collections of diversely owned machines. It utilizes a centralized scheduler based on the ClassAd matchmaker. To overcome the disadvantages of centralized scheduling, Condor allows the matchmaker (and/or the user) to forward requests to another matchmaker through the gateway flocking mechanism. The Condor project is still under development and has a large community of users.

International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-2, Issue-1,January 2016
ISSN: 2395-3470
www.ijseas.com

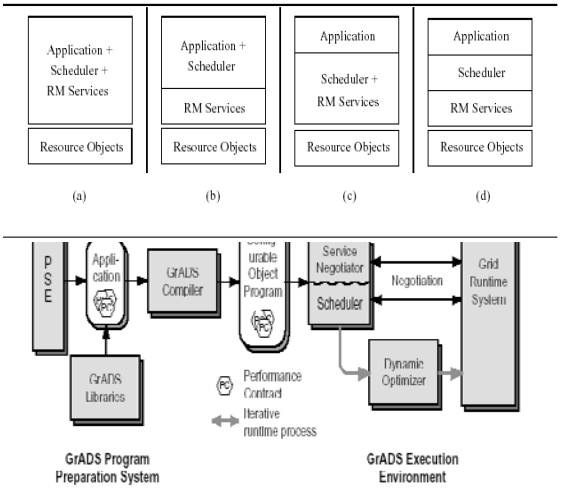


Community Scheduler Framework

The Community Scheduler Framework (CSF) is an open source Web Services Resource Framework compliant metascheduler built for the Globus Toolkit. The CSF provides interface and tools for Globus users to create reservations, define scheduling policies and submit jobs to the Grid. CSF functionalities can be extended to utilize other schedulers and support different Grid deployment models. For instance, using CSF allows a single interface access to (i) Load Sharing Facility (LSF) (ii) OpenPBS, (iii) Condor, and (iv) Sun Grid Engine (SGE). The CSF is the default metascheduler for the Globus Toolkit 4. This indicates that the CSF is not only active, but likely to be developed further (with the development of Globus).

GraDS: Grid Application Development Software (GRaDS) Project with support from NSF Next Generation Software Program has been developing tools for construction of applications on the grid easier. This led to the development of a prototype software infrastructure called GrADSoft that runs on top of Globus and facilitates scheduling, launching and performance monitoring of tightly coupled Grid applications. In GrADS, the end user just submits their parallel application to the framework for execution. The framework schedules the application to appropriate set of resources, launching and monitoring the execution and also rescheduling the applications on different set of resources if necessary. Anirban Mandal et al has launched and executed EMAN, a Bio imaging workflow application onto the grid . F.Berman et al in has presented an extension to GrADS software framework for scheduling workflow computations that has been applied to a 3-D image reconstruction application etc. [18].

Legion: Legion is an object based, meta systems software project at the University of Virginia that began in late 1993. It has been created to address key issues such as scalability, programming ease, fault tolerance, site autonomy, security etc. It was also designed to support large degrees of parallelism in application code and manage the complexities of the physical system for the user. It also allows applications developers to select and define system-level responsibility. Anand Natrajan et al in [8] has presented a grid resource management of legion for scheduling all compute objects as well as data objects on machines whose capabilities match the requirements, while preserving site autonomy as well as recognizing



usage policies.

NetSolve

The NetSolve system from the University of Tennessee's Innovative Computing Laboratory was to address the ease of use, portability and availability of optimized software libraries for high performance computing. It enables users to solve complex scientific problems remotely, by managing networked computational resources and using scheduling heuristics to allocate resources to satisfy the requests . The Primary goal of NetSolve was to make an easy access to grid resources. NetSolve, which is a client-agent server system provides remote access to hardware as well as software resources. The agent maintains a list of all available severs and performs resource selection for client requests and also ensures load balancing of the servers. Even though locating appropriate resources to the request is a challenge in grid computing, the NetSolve agent uses knowledge of the requested service, information about the parameters of the service request from the client, and the current state of the resources to get possible servers and return the servers in sorted order .

Sun Grid Engine : Sun Grid Engine (SGE) is the foundation of Sun Grid Utility Computing system, made available over Internet in the United States in 2006, later available in many other countries. It is used on high performance computing cluster is used for accepting, scheduling, dispatching and managing remote and distributed execution of large numbers of standalone or parallel user jobs. It also schedules the allocation of distributed resources such as processors, memory, disk space etc. Some of the features of SGE include advance reservation of resources, multi clustering, job submission verifier on both client and server sides, topology aware scheduling, job and scheduler fault tolerance etc. Goncalo Borges et al in has presented a work developed to integrate SGE with the EGEE (Enabling Grids for E-Science) middleware. EGEE is the world's largest operating grid infrastructure serving thousands of multi science users with robust, reliable and secure grid services worldwide.

Types of Scheduling in Grids

Scheduling is a family of problems: on the one hand, different applications could have different scheduling needs such as batch or immediate mode, task independent or dependent; on the other hand, the Grid environment characteristics itself imposes restrictions such as dynamics, use of local schedulers, centralized or decentralized view of the system, etc. It is clear that in order to achieve a good performance of the scheduler, both problem specifics and Grid environment information should be "embedded" in the scheduler. In the following, main types of scheduling arising in Grid environments is described.

Independent Scheduling

Computational Grids are parallel in nature. The potential of a massive capacity of parallel computation is one of the most attractive characteristics of the computational grids. Aside from the purely scientific needs, the computational power is causing changes in important industries such as biomedical one, oil exploration, digital animation, aviation, in financial field, and many others. They also appear in intensive computing applications and data intensive computing, data mining and massive processing of data, etc. The common characteristic in these uses is that the applications are written to be able to be partitioned into almost independent parts (or loosely coupled). For instance, an application of intensive use of CPUs can be thought of as an application composed by subtasks (also known as bags-of-tasks applications in Grid computing literature), each one capable to be executed in a different machine of the Computational Grid. This kind of applications require independent

scheduling, according to the following scenario: the tasks being submitted to the grid are independent.

Grid workflows

Solving many complex problems in Grids require the combination and orchestration of several processes (actors, services, etc.). This arises due to the dependencies in the solution flow (determined by control and data dependencies). This class of applications are know as Grid workflows, which can take advantage of the power of Grid computing.The scheduling terminology used by the Grid community seems quite different from the one used by the traditional scheduling community. Some basic definitions are listed out,which may clarify or expand the Grid scheduling terminology:

Immediate mode scheduling: In the immediate mode scheduling, tasks, jobs or applications are scheduled as soon as they enter the system.

Batch model scheduling: In the batch mode scheduling, tasks, jobs or applications are grouped into batches which are allocated to the resources by the scheduler. The results of processing are usually obtained at a later time.

Non-preemptive/Preemptive scheduling: This classification of scheduling establishes whether a task, job or application can be interrupted or not, once allocated to the resource. In the non-preemptive mode, a task, job or application should entirely be completed in the resource (the resource cannot be taken away from the task, job or application). In the preemptive mode, the preemption is allowed, that is, the current execution of the job can be interrupted and the job is migrated to another resource. Preemption can be useful if job priority is to be considered as one of the constraints.

Real Time Scheduling: Real-Time scheduling problems are usually physically or functionally distributed (air traffic control, manufacturing systems, health care, etc.).Complex scheduling systems are beyond direct control. They operate through the co-operation of many interacting subsystems, which may have their independent interest, and modes of operation. The complexity of practical scheduling problems dictates a local point of view. When the problems are too extensive to be analyzed as a whole, solutions based on local approaches are more efficient. There is a need for integration of multiple legacy systems and expertise. Real-world scheduling problems are heterogeneous. Heterogeneous environments may use different data and models, and operate in different modes.

Phases of scheduling in Grids

In order to perform the scheduling process, the Grid scheduler has to follow a series of steps which could be classified into five blocks: (1) Preparation and information gathering on tasks, jobs or applications submitted to the Grid; (2) Resource selection; (3) Computation of the planning of tasks (jobs or applications) to selected resources; (4) Task (job or application) allocation according to the planning (the mapping of tasks, jobs or applications to selected resources); and, (5) Monitoring of task, job or application completion (the user is referred to [61] for a detailed description). Preparation and information gathering: The Grid scheduler will have access to the Grid information on available resources and tasks, jobs or applications (usually known as “Grid Information Service” in the Grid literature). Moreover, the scheduler will be informed about updated information (according to the scheduling mode). This information is crucial for the scheduler in order to compute the planning of tasks, jobs or applications to the resources.

Computation of the planning of tasks: In this phase the planning is computed.

Task allocation: In this phase the planning is made effective: tasks (jobs or applications) are allocated to the selected resources according to the planning.

Task execution monitoring: Once the allocation is done, the monitoring will inform about the execution progress as well as possible failures of jobs, which depending on the scheduling policy will be rescheduled again (or migrated to another resource).

Different stages of Grid Scheduling : Grid scheduling involves four main stages. They are,

- Resource discovery,
- Resource selection,
- Schedule generation
- and job execution.

Resource discovery

The goal of resource discovery is to identify a list of authenticated resources that are available for job submission. Resource availability is determined in-terms of no of registers and functional units available, memory units, cache configuration, available bandwidth ,CPU usage etc..To adapt with the dynamic nature of the Grid, a scheduler needs to know some way of incorporating dynamic state information about the available resources into its decision-making process. Similarly, a scheduler should always know what resources it can access, how busy they are, how long it takes to communicate with them and how long it takes for them to communicate with each other. With this information, the scheduler optimizes the scheduling of jobs to make more efficient and effective use of the available resources.

Schedule generation

The generation of schedules involves two steps, selecting jobs and producing resource selection strategies.

Resource selection

Resource selection is the second phase of the scheduling process which selects resources that best suit the constraints and conditions imposed by the user, such as CPU usage, RAM available or disk storage. The result of resource selection is to identify a resource list which meets the minimum requirements for a submitted job or a job list.

The goal of job selection is to select a job from a job queue for execution. Four strategies that can be used to select a job are given below.

- First come first serve
- Random selection
- Priority-based selection
- ***Priority Queuing:*** Priority Queuing has two sub types such as Head of Line (HOL) for a fixed priority and Dynamic Queue (DQ) where each arriving job is allocated a priority. Its priority level determines the position of the job in the queue.

1. **Three Issues:** While implementing priority queuing, two issues have to be considered to avoid pitfalls.

2. Starvation: When high priority processes continuously enter the queue, lower priority processes may not be able to get processing time. If pre-emption is allowed, even if a low priority process were to start getting processed, it may be pushed out as soon as a high priority process was to come in. In such a situation, the lower priority processes are said to face starvation.

3. **Solution:** Increase the priority level of a process depending upon the waiting time in the queue of the process. Thus over a period of time an initially low-priority process may be able to beat a high-priority process, which has just entered the queue.

4. Deadlocking: A high priority process may need some resource, which can be created only when a lower priority process is executed. However since the high priority process is supposed to be processed first, a deadlock may occur, in that neither the high priority nor the low priority process may be executed.

5. Solution: Priority Inversion: The priority of the lower-priority process may be temporarily boosted so that it may be executed, before the high priority task is processed.

6. Backfilling selection- refers to the case, where a task at the back in the Wait queue, is chosen on the basis of some criterion to be processed before the first task in the Wait queue is taken up for processing. Sometimes when a compute node becomes free, it may not be possible to allocate the first task in the wait queue due to dependencies among the tasks. In such a case rather than keeping the node idle, an out-of-order task may be allocated to the free node. Two common variants are Easy and Conservative backfilling. In Conservative backfilling, the out-of-order task must be chosen such that the first job in the Wait queue is not delayed. "The performance of this algorithm depends upon a sufficiently large backlog."

7. Gang Scheduling: In this approach the main concept is to "add a time-sharing dimension to space sharing using a technique called gang scheduling or co scheduling. This technique virtualizes the physical machine by slicing the time axis into multiple virtual machines. Tasks of a parallel job are co-scheduled to run in the same time-slices (same virtual machines) "

8. Genetic Algorithms (GAs): GAs are useful for optimization problems, in cases where the number of parameters, which affect the performance are large. GAs begins with a potential set of solutions, called the population of the search space and a fitness function, appropriately defined for the problem on hand. GAs then try to obtain an optimum solution by using the processes of cross-over and mutation. "GAs are widely reckoned as effective techniques in solving numerous optimization problems because they can potentially locate better solutions at the expense of longer running time. Another merit of a genetic search is that its inherent parallelism can be exploited to further reduce its running time" [11].

- **Directed Acyclic Graphs Scheduling:** A job, which is to be run on a parallel system of computational nodes, can be represented by a weighted-node and weighted-edge DAG. The weight of a node is the time that it would take to be processed. The directed edges define the dependencies. The weight of the edges may represent the communication delays."The objective of DAG scheduling is to minimize the overall program finish-time by proper allocation of the tasks to the processors and arrangement of execution sequencing of the tasks. Scheduling is done in such a manner that the precedence constraints among the program tasks are preserved. The overall finish-time of a parallel program is commonly called the schedule length or makespan" [11].

Job execution

Once a job and a resource are selected, the next step is to submit the job to the resource for execution. Job execution may be as easy as running a single command or as complicated as running a series of scripts that may, or may not, include set up or staging.

Properties of a Good Scheduling System

A good scheduling system on the Grid should have the following features as cited in[3] is as follows:

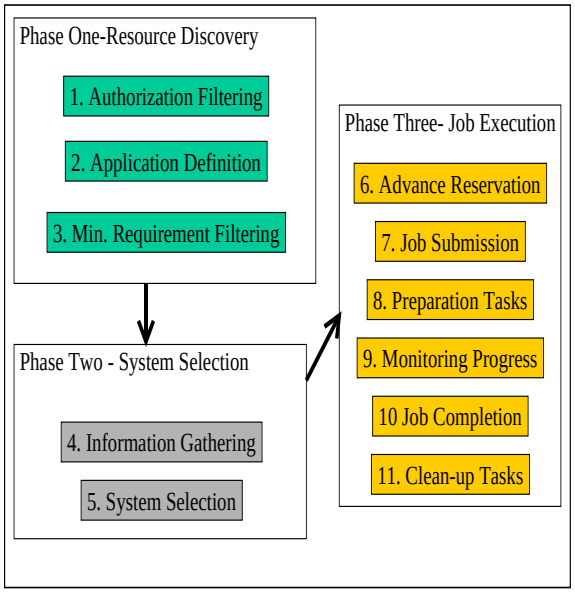
1. Efficiency in generating schedules
2. Adaptability, where resources may join or leave dynamically

3. Scalability in managing resources and jobs

Ability to predict and estimate performance

1. Ability to coordinate the competition and collaboration of different
2. Cluster-level schedulers
3. Ability to reserve resources for scheduling
4. Ability to take the cost of resources into account when scheduling
5. Ability to take user preferences and site policies into account.

Activities of a Grid Scheduler



Step 1: Authorization Filtering

The first step of resource discovery for Grid scheduling is to determine the set of authorized resources that the user submitting the job has access to: without authorization to run on a resource, the job will not run cited in [3].

Step 2: Application Requirement Definition

In this step, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources .The set of possible job requirements can be very broad and it may include static details like the operating system or hardware for which a binary of the code is available, or the specific architecture for which the code is best suited as well as dynamic details like, a minimum RAM requirement, connectivity needed, or /tmp space needed etc..[3]

Step 3: Minimal Requirement Filtering

In the Minimal Requirement Filtering phase, the resources that do not meet the minimal job requirements are filtered out.

Phase 2: System Selection

Given a group of possible resources all of which meet the minimum requirements for the job, a single resource must be selected on which to schedule the job. This selection is generally done in two steps: gathering detailed information and making a decision.

Step 4: Dynamic Information Gathering

In order to make the best possible job/resource match, detailed dynamic information about the resources is needed. The dynamic information gathering step has two components: what information is available and how the user can get access to it.

Step 5: System Selection

With the detailed information gathered in Step 4, the next step is to decide which resource (or set of resources) to use. Various approaches are , Condor matchmaking , multi-criteria , and meta-heuristics.

Phase 3: Job Execution

The third phase of Grid scheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources.

Step 6: Advance Reservation (Optional)

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means resources.

Step 7: Job Submission

Once resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated as running a series of scripts and may or may not include setup or staging.

Step 8: Preparation Tasks

The preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application.

Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and the monitoring is typically done by repetitively querying the resource for status information, but this is changing over time to allow easier access to the data. If a job is not making sufficient progress, it may be rescheduled.

Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter. For fault-tolerant reasons, however, such notification can prove surprisingly difficult. And of course, end-to-end performance monitoring to ensure job completion is a very open research question.

Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Users generally do this by hand after a job is run, or by including clean-up information in their job submission scripts.

Functional Requirements for Grid Scheduling

- Cooperation between different Resource providers
- Interaction with Local Resource Management systems
- Support for reservations and service level agreements
- Orchestration of coordinated resources allocations
- Automatic handling of accounting and billing
- Distributed Monitoring
- Failure Transparency
- Resource Management in a Multicore Grid

Scheduling algorithms can be distinguished by their main characteristics, such as:

- **Target system:** the system for which the scheduling algorithm was developed, which can be a heterogeneous system, a grid, or a cloud computing system.
- **Optimization criterion:** Make span and cost are the main metrics specified by cloud user and considered by schedulers in the decision making process.
- **Multi-core awareness:** Computer systems can have multiple cores which should be considered by scheduling algorithms in resource selection.
- **On-demand resources:** Resources can be leased either on-demand or for long terms. The on-demand leasing of resources is treated by the scheduling

algorithm as a “single expense” during the execution of the workflow.

Reserved resources: The algorithm should consider the use of a resource reserved for a long term.

Levels in service level agreement (SLA): The scheduling algorithm should consider that SLAs can be organized hierarchically. SLAs with a single-level allow clients and providers to interact directly to negotiate resource capacities and prices. When multiple levels, the scheduling algorithm can run in an intermediate facility between the IaaS cloud provider and the final client. By doing so, costs can be decreased.

A review of traditional scheduling Methodologies

Grid scheduling could benefit from several traditional scheduling methodologies. These methodologies have achieved successful results in a wide range of scheduling applications. Therefore, it worth to start to investigate their performance in Grid scheduling. These methodologies are described below.[5]

Heuristics:

A Heuristic is a technique that seeks good solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is [Reeves, 1995]. Dispatching rules are example of heuristics; they are used to select the next job to process on the resource whenever the resource becomes free. Dispatch rules include EDD (Earliest Due Date) and FCFS (First Come First Served).[5]

Meta-heuristics: tabu search, simulated annealing and evolutionary algorithms.Meta-heuristics are high-level heuristics that guide local search heuristics to escape from local optima. Meta-heuristics such as tabu search, simulated annealing and genetic algorithms improve the local search algorithms to escape local optima by either accepting worse solutions, or by generating good starting solutions for the local search in a more intelligent way than just providing random initial solutions [Reeves, 1995, 1997; Reeves and Rowe, 2002; Voss et al., 1999; Pham and Karaboga, 2000].

Case-based reasoning

Case-Based Reasoning (CBR) is an artificial intelligence methodology in which a new problem is solved by reusing knowledge and experience gained in solving previous problems. A case contains a description of the problem, and its solution. Cases are stored in a case base. The CBR process is divided into four phases: retrieval of the case most similar to the new problem, reuse and revision of its solution, and inclusion of the new case in the case base [Kolodner, 1993; Aamodt and Plaza, 1994; Leake, 1996].

Dynamic scheduling

Dynamic scheduling is the problem of scheduling in dynamic environments. Grid scheduling systems operate in dynamic environments subject to various unforeseen and unplanned events that can happen at short notice. Such events include the breakdown of computers, arrival of new jobs, processing times are subject to stochastic variations, etc. It turns out that the performance of a schedule is very sensitive to these disturbances, and it is difficult to execute a predictive schedule generated in advance. These real-time events not only interrupt system operation but also upset the predictive schedule that was previously established. Consequently the resulting schedule may neither be feasible nor nearly optimal anymore. Dynamic scheduling is arguably of practical importance in Grid Scheduling to generate robust schedules. For extensive surveys on dynamic scheduling refer to [Cowling and

Johanson, 2002; Cowling et al., 2003a; Vieira et al., 2003].

Fuzzy methodologies

Fuzzy systems consist of a variety of concepts and techniques for representing and inferring knowledge that is imprecise, uncertain, or unreliable. fuzzy set is a very general concept that extends the notion of a standard set defined by a binary membership to accommodate gradual transitions through various degrees. Previous work has investigated the representation of uncertainty in processing time and due time by fuzzy numbers, the representation of flexible constraints by fuzzy measures, fuzzy job precedence relations or machine breakdowns, but these have been in isolation. Even once an SLA has been agreed, there are many ways in which it might need renegotiation: (compute and other) resources may fail unpredictably, sub-jobs may fail due to user error, more important (high-priority) jobs may be submitted, user-requirements might change, etc. In a busy Grid environment, SLAs would be constantly being added, altered or withdrawn, and hence scheduling would need to be a continual, dynamic and uncertain process.

Agents and multi-agent systems

Recently, multi-agent systems are one of the most promising approaches to building complex, robust, and cost-effective next-generation manufacturing scheduling systems because of their autonomous, distributed and dynamic nature, and their robustness against failures. An agent is a computer system that is situated in some environment, and that is capable of flexible and autonomous action in this environment in order to meet its design objectives.

A Multi-Agent System is a system composed of a population of autonomous agents, which interact with each other to reach common objectives. Agents provide robustness and reliability against failures. Distributed systems allow fast detection and recovery from failures and the failure of one or several agents does not necessary make the overall system useless. Because multi-agent systems are open and dynamic structures, the system can be adapted to an increased problem size by adding new agents, and this does not affect the functionality of the other agents. Agents can operate asynchronously and in parallel, which can result in increased overall speed. Individual agents can be developed separately and it may be possible to reuse agents in different application scenarios. Moreover, the overall system can be tested and maintained and reconfigured more easily. Agents may be much more cost-effective than a centralized system, since it could be composed of simple subsystems of low unit cost. Multi-agents have been successfully used to resolve a wide range of complex distributed scheduling problems . We believe that Multi-agent systems provide the foundation for the creation of Grid scheduling systems that possess capabilities of autonomy, heterogeneity, reliability, maintainability, flexibility, and robustness. Some researchers have already started to investigate the use of the agent technology in grid computing, in particular for resource management in grid environments [Cao et al., 2002a, 2002b].

Literature Survey of Traditional Scheduling Algorithms

First come first served scheduling algorithm (FCFS)

In this algorithm, jobs are executed according to the order of job arriving time. The next job will be executed in turn. The FCFS algorithm [1] may induce a ‘‘convoy effect’’. The convoy effect happens when there is a job with a large amount of workload in the job queue. When this occurs, all the jobs queued behind it must wait a long time for the long job to finish.[17.22,24]

Round Robin scheduling algorithm (RR)

The RR algorithm mainly focuses on the fairness problem. The RR algorithm [2] defines a ring as its queue and also defines a fixed time quantum. Each job can be executed only within this quantum, and in turn. If the job cannot be completed in one quantum, it will return to the queue and wait for the next round. The major advantage of RR algorithm is that jobs are executed in turn and do not need to wait for the previous job completion. Therefore, it does not suffer from a starvation problem. However, if the job queue is fully loaded or workload is heavy, it will take a lot of time to complete all the jobs. Furthermore, a suitable time quantum is difficult to decide.[17,22,24]

Min–min and Max–min algorithm

The Min–min scheduling algorithm [3] sets the jobs that can be completed earliest with the highest priority. Each job will always be assigned to the resource that can complete it earliest. Similar to Min–min algorithm, Max–min algorithm [3] sets the highest priority to the job with the maximum earliest completion time. The main idea of Max–min algorithm is to overlap long running tasks with short-running tasks. Max–min can be used in cases where there are many shorter tasks than there are longer tasks. For example, if there is only one long task, Min–min will first execute many short jobs concurrently, and then execute the long task. Max–min will execute short jobs concurrently with the long job.

Sufferage scheduling algorithm

The idea behind the sufferage scheduling algorithm is that better mapping can be generated by assigning a machine to a task that would ‘‘suffer’’ most in terms of expected completion time if that machine is not assigned to it. In this algorithm, each job is assigned according to its sufferage value. The sufferage value is defined as the difference between its second earliest completion time and its earliest completion time (two completion times with different resources). The sufferage algorithm will pick a job in an arbitrary order and assign it to the resource that gives the earliest completion time. If another job has the earliest completion time with same resource, the scheduler will compare their sufferage values and choose the larger one. However, this algorithm may have the starvation problem.

On-line mode heuristic scheduling algorithm

Jobs are scheduled when they arrive. Since the Grid environment is a heterogeneous system and the speed of each processor varies quickly, the on-line mode heuristic scheduling algorithms are more appropriate for the Grid environment.

Most fit task scheduling algorithm (MFTF)

The MFTF algorithm [9] mainly attempts to discover the fitness between tasks and resources for user. It assigns resources to tasks according to a fitness value, and the value is calculated as follows:

$$\text{fitness}(i, j) = \frac{10000}{1+(Wi/Sj-Ei)}$$

where Wi is the workload of the ith task, Sj is the CPU speed of the jth node, and Ei is the expected time of the ith task. Wi/Sj is the expected execution time using this node|Wi/Sj–Ei| is the difference of the estimated execution time and the expected task execution time. Ei is determined by the user or estimated by the machine. How to set Ei is calculated by, Ei = A + n × S, where A is the average response time of the 100 latest done tasks;n is a non-negative real number and S is the standard deviation of task response time for the 100 latest done task. When the estimated execution time is closer to Ei, it means that the node is more suitable for the task. However, the MFTF scheduling algorithm has some problems for estimating. It does not

consider the resource utilization, and the estimated function is an ideal method. Therefore, incorrect scheduling may occur in the real environment.

Ant algorithm

The ant algorithm is also based upon heuristic approach. It is based on the behavior of real ants. Each ant deposes the chemical pheromone on its path when it searches for food fromits nest. When each ant moves in a particular direction, the strength of chemical pheromone increases. With this, other ants could also trail along. This inspired the discovery of ACO algorithm. This algorithm uses a colony of artificial ants that behave as cooperative agents in a mathematical space where they are allowed to search and reinforce pathways (solutions) in order to find the optimal ones(i.e) the shortest path. This approach which is population based has been successfully applied to many NP-hard optimization problems [3][4].

Ant algorithm-based task scheduling in grid computing

Ant algorithm is a new heuristic algorithm; it is based on the behavior of real ants. When the blind insects, such as ants look for food, every moving ant lays some pheromone on the path, then the pheromone on shorter path will be increased quickly, the quantity of pheromone on every path will effect the possibility of other ants to select path. At last all the ants will choose the shortest path. Ant algorithm has been successfully used to solve many NT' problems, such as TSP, assignment problem, job-shop scheduling and graph coloring. The algorithm has inherent parallelism, and we can validate its scalability. So it's obvious that ant algorithm is suitable to be used in Grid computing task scheduling. And at the same time, all the factors that affects the state of resources can be described by one thing, pheromone. Then we can get the predictive results very simple and quickly.

An Improved Ant Algorithm For Job Scheduling In Grid Computing

In this paper, we propose an improved ant algorithm for job scheduling in grid computing. The new algorithm is based on the general ant adaptive scheduling heuristics and an added in load balancing guide component. The load balancing factor, related to the job finishing rate, is introduced to change the pheromone. That will make the job finishing rate at different resource being similar and the ability of the systematic load balancing will be improved. It has been successfully tested in a simulation grid environment. The experiments show that the new ant heuristic method can lead to significant performance in various applications.[17,22,24]

Ant colony algorithm

The ant colony algorithm for job scheduling in grid aims at submitted jobs to resources based on the processing ability of jobs as well as the characteristics of the jobs. Ant colony algorithm is the bio-inspired heuristic algorithm, which is derived from the social behavior of ants. Ants work together to find the shortest path between their nest and food source. When the ants move, each ant will deposit a chemical substance called pheromone. Using this pheromone, the shortest path is found. The same concept is used to assign jobs in grid computing. When a resource is assigning a job and completes, its pheromone value will be added each time. If a resource fails to finish a job, it will be punished by adding less pheromone value. The issue here is the stagnation, where there is a possibility of jobs being submitted to same resources having high pheromone value.In this ant colony algorithm [18], the load balancing method is proposed

to solve the issue of stagnation. The algorithm is as follows

(i) The user will send request to process a job.

(ii) The grid resource broker will find a resource for the job.

(iii)The resource broker will select the resource based on the largest value in the pheromone value matrix.

(iv)The local pheromone update is done when a job is assigned to a resource.

(v)The global pheromone update is done when a resource completes a job.

(vi)The execution result will be sent to the user when the resource broker select a particular resourcefor a job j, jthcolumn of the Pheromone Value matrix will be removed and jobs will be assigned to other resources. Thus the load balancing is achieved.

Ant colony optimization (aco) algorithm in job scheduling

ACO has been used to solve scheduling problems in the Grid environment in recent years [10]. ACO algorithm is based on Ant algorithm and modified it to suit the Grid environment. Like ACO algorithms, they needed some information such as number of CPUs, MIPS for each processor, etc. to schedule tasks. They used a parameter named pheromone to do the scheduling action. A resource must submit the information to the resource monitor, and the pheromone values are initialized at the beginning of the algorithm [20].

However, the better resources will get more load capacity than the others, and the performance will decrease. The load balancing factor means that the more jobs finished in a resource, the more its pheromone intensity increases. In contrast, the more jobs failed, the more pheromone intensity is decreased. Therefore, they added the load balancing factor to reduce the load of the better resources to balance the utilization of the resources in the grid environment. However, the definition of the pheromone has a problem. The value is the sum of different units. This problem may affect the result of scheduling algorithm accuracy.

A New Ant Colony Optimization Scheduling Algorithm : The proposed scheduler proves that best suitable resource is allocated to each task with reduced makespan and execution time when compared with the existing algorithm. This algorithm gives the efficient resource allocation to the machines. It is working with only resource allocation and not on other criteria such as minimizing completion time, execution time etc.

Particle Swarm Optimization (PSO) is a population based search algorithm. The particles fly through a multidimensional search space in which the position of each particle is adjusted according to its own experience and the experience of its neighbors. In the binary version of this algorithm was presented by Kennedy and Eberhart [10] ,In which, each particle is composed of D elements, which indicate a potential solution. In order to evaluate the appropriateness of solutions a fitness function is always used. Each particle is considered as a position in a D dimensional space and each element of a particle position can take the binary value of 0 or 1 in which 1 means “included” and 0 means “not included”. Each element can change from 0 to 1 and vice versa. Also each particle has a D-dimensional velocity vector the elements of which are in range[-Vmax ,Vmax] .Velocities are defined in terms of probabilities that a bit will be in one state or the other. The velocity vector is updated in each time step using two best positions, *pbest* and *nbest* , and then the position of the particles is updated using velocity vectors. *Pbest* and *nbest* are D dimensional, the elements of which are composed of 0 and 1 the same as particles position and operate as the memory of the algorithm. The personal best position, *pbest* , is the best position the particle has visited and

nbest is the best position the particle and its neighbors have visited since the first time step. When all of the population size of the swarm is considered as the neighbor of a particle, *nbest* is called global best (star neighborhood topology) and if the smaller neighborhoods are defined for each particle (e.g. ring neighborhood topology), then *nbest* is called local [9].

Heuristic min-mean job scheduling page layout

Online mode and batch mode are the two classifications of heuristic scheduling algorithms [9] [10]. In online mode, the jobs are scheduled to the resources as soon as it arrives. In batch mode, the jobs are independent, there is no order of execution and jobs are scheduled as a batch every time. Here in this paper, batch mode scheduling is followed. Achieving the minimum make span is the goal. To evaluate the mapping heuristic, the expected time to complete [ETC]model is employed. Before execution, the expected execution time of the tasks on the machine should be known, and this is contained in the ETC matrix. Consider for the task ti and the arbitrary machine mjETC[ti,mj]. This represents the expected time of the task i on the machine j. In this matrix, the row represents the expected execution time of a task on different machine and column represents the expected execution time of different tasks on the same machine[11][12][13]. Based on these characteristics, the benchmark of instances for distributed heterogeneous computing system is generated.

(i) Machine heterogeneity (low/high)

(ii) Task heterogeneity (low/high)

(iii)Consistency(Consistent/Inconsistent/Partially Consistent)

Combining these 3 bench marks, 12 combinations of ETC matrices are used to evaluate the heuristic min mean scheduling algorithm. There are 2 phases in this algorithm [14]. In the first phase, all jobs are assigned to the resources. In the second phase, mean completion time of all jobs is calculated and the jobs are allocated to the machines whose completion time is less than the mean completion time. Machine who has maximum completion time is selected as make span.

Firefly algorithm

The firefly algorithm is based on swarm intelligence behavior of firefly. It is a meta heuristic algorithm inspired by the social behavior of firefly. Firefly algorithm finds the global optimal solution. The main focus of firefly algorithm is to complete the task within a minimum make span and flow time as well to utilize the grid resource efficiently. Firefly optimization as mentioned in [15] [16] in can be described as

- The firefly attracts and is attracted by all other Fireflies
- The brighter one attracts the less bright one
- The brightness decreases with distance
- The brightest firefly can move randomly
- The firefly particles can move randomly

There are 4 phases in firefly algorithm [17] In the phase 1, the parameters are set (initial population, fitness and attractiveness), number of available resources and list of submitted jobs are identified. In the phase 2, the brightness of each firefly is found at the source using fitness function and distance is calculated. The less bright fireflies are moved towards the brighter one. In the phase 3, the new solution is evaluated and light intensity is updated In the phase 4, the fireflies are ranked and current global best is identified. Finally, the iteration parameters are updated All these are done until the termination condition is reached. The termination condition may be number of

iteration or the fitness value or sometimes the saturation state.

Grouping based job scheduling in grid computing

The adaptive group based job scheduling focuses on group based scheduling, and explains the jobs are grouped based on coarse grained jobs. The grouping is based on the resources' processing capability in Million Instructions per Second (MIPS),bandwidth (Mb/s), and memory size (In Mb).The characteristics of resources are the basic for grouping strategy. In grid computing, 2 approaches can be used for job execution. In first approach, the user can directly search the resources for job execution using an information service. In the second approach, the user can obtain information about the current availability and capability of resources using the resource manager. The Grouping Based Job Scheduling algorithm has 2 phases [19]. In the first phase, the scheduler receives information about the resource status from the Grid Information Service (GIS), and sorts the jobs in descending order. In the second phase, the system selects jobs in First Come First Served (FCFS) order and forms different job groups. The scheduler will select resource in FCFS order after sorting them in descending order of their MIPS. The jobs are put into the job groups until the sum of resource requirement of the jobs in that group is less than or equal to amount of resource available at selected site. As soon as the job group is formed, the jobs are assigned to the corresponding resource. After execution the job groups, the result is sent to the corresponding user and the resources are available to the Grid System. The .NET framework scheduler uses the FCFS scheduling technique.[7]

Community – aware scheduling algorithm (CASA)

CASA is a decentralized dynamic heuristic meta scheduling algorithm. In CASA, jobs can be rescheduled. In order to overcome the stagnation a probabilistic approach has been used to assign jobs so that the jobs are evenly distributed to all other resources. CASA is a two phase algorithm [20]. The first phase is the job submission phase where each node receives the jobs that are submitted by local user. Consider a node A, it receives the job, it acts as a initiator node and requests all other nodes using the REQUEST message. The other nodes who are willing to take the job will reply through ACCEPT message. The node A will evaluate the other participating nodes using the historic data and selects the appropriate node and submits the job to it. The second phase is the dynamic rescheduling phase,the node which received the job will look for the job which has large enough waiting time and has not been selected recently in the local job queue. That job will be rescheduled to the other nodes.5 algorithms are discussed in CASA. They are:

- Job distribution
- Job delegation request acceptance
- Job assignment
- Job rescheduling
- Job rescheduling request acceptance. [7]

MET (Minimum Execution Time): MET assigns each task to the resource with the best expected execution time for that task, no matter whether this resource is available or not at the present time. The motivation behind MET is to give each task its best machine. This can cause a severe load imbalance among machines. Even worse, this heuristic is not applicable to heterogeneous computing environments where resources and tasks are characterized as consistent, which means a machine that can run a task faster will run all the other tasks faster.

MCT (Minimum Completion Time): MCT assigns each task, in an arbitrary order, the resource with the minimum expected completion

time for that task. This causes some tasks to be assigned to machines that do not have the minimum execution time for them.The intuition behind MCT is to combine the benefits of opportunistic load balancing (OLB) and MET, while avoiding the circumstances in which OLB and MET perform poorly.

Min-min: The Min-min heuristic begins with the set *U* of all unmapped tasks. Then, the set of minimum completion time *M* for each task in *U* is found. Next, the task with the overall minimum completion time from *M* is selected and assigned to the corresponding machine (hence the name Min-min). Last, the newly mapped task is removed from *U*, and the process repeats until all tasks are mapped (i.e., *U* is empty). Min-min is based on the minimum completion time, as is MCT. However, Min-min considers all unmapped tasks during each mapping decision and MCT only considers one task at a time. Min-min maps the tasks in the order that changes the machine availability status by the least amount that any assignment could. Therefore, the percentage of tasks assigned to their first choice (on the basis of execution time) is likely to be higher for Min-min than for Max-min. The expectation is that a smaller makespan can be obtained if more tasks are assigned to the machines that complete them the earliest and also execute them the fastest.

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

XSuffrage: Another popular heuristic for independent scheduling is called *Suffrage* [76]. The rationale behind Suffrage is that a task should be assigned to a certain host and if it does not go to that host, it will suffer the most. For each task, its suffrage value is defined as the difference between its best MCT and its second-best MCT. Tasks with high suffrage value take precedence. But when there is input and output data for the tasks, and resources are clustered, conventional suffrage algorithms may have problems. In this case, intuitively, tasks should be assigned to the resources as near as possible to the data source to reduce the makespan. But if the resources are clustered, and nodes in the same cluster are with near identical performance, then the best and second best MCTs are also nearly identical which makes the suffrage close to zero and gives the tasks low priority. Other tasks might be assigned on these nodes so that the task might be pushed away from its data source. To fix this problem, Casanova et al gave an improvement called *XSuffrage* in [23] which gives a cluster level suffrage value to each task. Experiments show that XSuffrage outperforms the conventional Suffrage not only in the case where large data files are needed, but also when the resource information cannot be predicted very accurately.

Estimation Based Grid Scheduling Approach (EBGSA),allows for the simultaneous processing of grid tasks and local tasks. In EBGSA, expected execution time is treated as a random variable in stead of a predetermined constant. By estimating the value of each random variable, the scheduler can make a better schedule, which takes into account the actual resource status in the grid.

Efficient Utilization of Computing Resources Using Highest Response Next Scheduling in Grid (HRN), provides more responses with time, memory and CPU requirements. Here, jobs are allocated to N number of processors based on jobs priority and processors capability. This scheme is adaptive for local jobs and remote jobs without any loss of performance and also highly adaptive for grid environment. HRN with priority will effectively utilize the available resource and complete all the jobs quickly than FCFS. It corrects some of the weakness of both Shortest Job First (SJF) and First Come First Serve (FCFS) are some of the benefits of HRN but it is not suitable for more number of jobs allocations because finding priority of job is tedious one.

Node Allocation in Grid Computing Using Optimal Resource Constraint (ORC) Scheduling.

The ORC algorithm allocates the jobs according to processors capability. It applies best fit algorithm followed by Round Robin (RR) scheduling which distributes the jobs among the available processors. ORC is compared with different algorithms like FCFS, SJF and RR. The comparison shows that ORC gives better performance than other algorithms in terms of turn around time and average waiting time. It overcomes the problem of FCFS and HRN scheduling policy as it is suitable for more number of jobs. It helps to minimize the complexity of process allocation, reduces the turnaround time and average waiting time of jobs in the queue which in turn avoids starvation problem. But ,it has High communication overhead.

Hierarchical Job Scheduling for Clusters of Workstations (HJS): The scheduling model is based on a hierarchical approach using two level scheduling consisting of top level global scheduling and local level scheduling. The global scheduler uses single or separate queue for different type of jobs for scheduling with FCFS, SJF or First Fit (FF) policy. The local scheduler uses only one queue for all types of jobs with any one policy FCFS, SJF or FF. The function of global scheduler is matching of the resources requested by a job to those available in the participating clusters and to obtain the best utilization of the available clusters. The local scheduler is responsible for scheduling jobs to a specific resource. At both levels, the schedulers strive to maintain a good load balance. It tries to reduce overall turnaround time and maximize system utilization for high system loads. For high system loads it uses multi queue to maintain the delay of job scheduling at global level.

Resource Co-allocation for Scheduling Tasks with Dependencies in grid (RCSTD): The Co-allocation scheduling algorithm provides a strategy for scheduling the tasks with dependencies in grid environment. The algorithm applies on both inside and across the clusters. Every step combines or merge the clusters (tasks inside the cluster or clusters across the cluster) based on the dependencies between the combined clusters. Thus these clusters are combined if any dependencies exist between current and previous cluster. The main goal of the algorithm is to improve efficiency in terms of load balancing and minimum time for the execution of the task. The Algorithm Minimizes Execution Time of the Task,it is dynamic in nature because inside a cluster the tasks are allocated to the suitable resource on

which it can be scheduled at the earliest time. Due to the decentralized strategy that Co-allocation uses, the method is more reliable than a centralized one for being less subject to single point of failure. This scheduling algorithm obtains good load balancing among all the resources of the system in terms of number of tasks scheduled on each resource. But more communication overhead occurs inside and across the clusters and no specified requirements of a task.

Research on Novel Dynamic Resource Management and Job Scheduling in grid computing (RNDRM): This scheduling model is based on Heap Sort Tree (HST)for computing the available computational power of the nodes (resource) as well as whole grid system. Here the resource with largest available computational ability among the whole grid system is selected to be the root node of the HST and it is ready for the scheduler to submit a job. The algorithm designed for job scheduling is well suitable for the complex grids environment and it is based on agents. This algorithm makes the system more scalable, robust, fault-tolerant and high performance. This strategy provides dynamic status information of the resources in an unpredictable fast changing grid environment. This algorithm is silent at the condition of job submission failure. The job scheduling strategy may not utilize resource sufficiently. Job waiting time is high and it does not provide real time dynamic grid environment.

Agent Based Resource Management with Alternate Solution (ABRMAS).

Agent based Resource Management with Alternate Solution gives an alternate solution at the situation when resource discovery fails. Algorithm identifies an equivalent resource without affecting the performance and it also avoids unnecessary resource discovery. Sometimes resource discovery is done for time bound task and required resource is unavailable at that situation. Alternate solution reduces delay overhead in waiting for the unavailable resource and enhances the systems efficiency. Implementation result shows the system success rate is 30% higher with alternate solution. It limits and steer the search towards the anticipated result and provide efficient resource discovery. Useful in both cases when discovery fails and more than one solution proposal offered. For large agent hierarchy proposal„s invitations may be restricted to sub hierarchy. It is not explicit.

New Resource Mechanism with Negotiate Solution based on agent in grid environments (NRMNS): Agent Based Resource Management with Negotiate Solution gives an alternate solution at the situation of resource discovery failure. Algorithm adds the middleware Grid Architecture for Computational Economy (GRACE) with Resource Pricing Fluctuation Manager (RPFM) into ABRMAS in order to improve the efficiency of the resource management scheduling allocation in Grid Computing. The feedback model plays a very important role in the agent-based system when resource discovery failed for cost bound. The resource provider can get the maximum investment profit. Feedback capability of RPFM is used to adapt the highly dynamic grid environment. Simulation result shows successful rate of resource discovery increases by about 10%.One of the major drawback is the resource discovery is aborted when the RPA (Resource provider agent) refuses to decrease the cost of the resource.

Improved Resource discovery approach using P2P model for condor (IRP2P): IRP2P is a grid middleware. It is a decentralized technique which opposes traditional client - server model. Goal of the model is to improve performance of condor middleware. Proposed hybrid model uses four axis frameworks in P2P approach. Each framework overcomes some limitations of

condor middleware and makes it more reliable, robust and scalable. By implementing membership protocol, network communication is easy and using overlay construction algorithm inter crosses communication is also allowed which is restricted in condor. Independence from central global control. Fast discovery of resources using DHTs and indexing concept. Supports Scalability and intermittent resource participation. Need to have strong self-organization capabilities in order to be able to maintain their rigid structure. High maintenance cost in the presence of high churn.

Virtual Computing Grid using Resource Pooling (VCGRP): The System is based on loosely coupled concept. Virtual Computing Grid means the system can choose a resource and allocate tasks to it. Here, it is a single point web based access known as Virtual Computing Grid Portal and the Virtual Computing Grid Monitor is a central resource manager for the System. It is a Cost Effective model. Not much Reliable because of only one central manager and single point web access.

Task Grouping: The above algorithms are usually used to schedule applications that consist of a set of independent coarse-grained compute-intensive tasks. This is the ideal case for which the computational Grid was designed. But there are some other cases in which applications with a large number of lightweight jobs. The overall processing of these applications involves a high overhead cost in terms of scheduling and transmission to or from Grid resources. Muthuvelu et al [79] propose a dynamic task grouping scheduling algorithm to deal with these cases. Once a set of fine grained tasks are received, the scheduler groups them according to their requirements for computation (measured in number of instructions) and the processing capability that a Grid resource can provide in a certain time period. All tasks in the same group are submitted to the same resource which can finish them all in the given time. By this mean, the overhead for scheduling and job launching is reduced and resource utilization is increased.

Existing Methods of Task Scheduling

Resource Aware Scheduling Algorithm - RASA is built through the analysis of two task scheduling algorithms, Min-min and Max-min and To achieve the lower make span this scheduling algorithm is used. RASA uses the advantages of Max-min and Min-min algorithms and covers their disadvantages. To minimize the make span. Applying the RASA algorithm on actual grid environment for practical evaluation can be open problem in this area.

Cuckoo Algorithm- A cuckoo optimization algorithm is proposed for optimal job allocation of resources on each node. This system will allocate the job optimally by considering the requirement of the users and also the minimal execution time.This system will allocate the job optimally by considering the requirement of the users and also the minimal execution time. Once job is allocated to the resource, it is fixed and no other way to change.

Load Balanced Min Min Algorithm (LBMM) algorithm is proposed to reduce the makespan and increases the resource utilization.LBMM uses the advantages of Max-min and Minmin algorithms and covers their disadvantages.To minimize the make span and increase the resource utilization**.**Applying the proposed algorithm on actual grid environment and considering the cost factor can be open problem in this area.

Qos Guided Weighted Mean Time Algorithm Resource load balancing and minimizing makespan are the fundamental goals of effective and efficient task scheduling. It turns out to be more complicated when various QoS demands arise from users. To perform the effective and efficient task scheduling this proposed system is used.Both heuristics QoS Guided Weighted

Mean Time min and QoS Guided Weighted Mean Time Min-Min .Max-Min Selective provide better makespan, resource utilization and load balancing than the other heuristics.It is only based on Quality of Service, the task can be divided into low and high Qos and schedules the task by means of high and low Qos.

Firefly Intelligent Swarm Optimization Technique : The proposed method is to dynamically create an optimal schedule to complete the tasks within minimum makespan. Complete the tasks within a minimum makespan and flowtime.This method concentrates only on completing the task within a minimum make span and not on other process such as resource allocation etc.

Multiple Ant Colony Optimization: The improved MACO approach is to find the optimal solution with a minimum execution time of task. Makespan is minimized by using the load balancing.

A Novel Qos Guided Task Scheduling Algorithm : The goal of grid task scheduling is to achieve high system throughput and to match the application needs with the available computing resources. Achieving high system throughput

A Probabilistic Task Scheduling Method : Probabilistic task scheduling method is used to minimize both the overall mean response time of the tasks which are submitted to the grid environments and total makespan of the environment. The overall mean response time is the important Quality of service measure in grid and to achieve this, Discrete Time Markov Chain is constructed. On absorbing all the DTMCs the Nonlinear programming problem is defined and by solving the NLP problem the best scheduling path and the minimum mean response time of a particular task can be obtained. Minimizing the overall mean response time, Minimizing the makespan of the job.

Issues in Computational grids

Scheduling Problems in Computational Grids

Rather than a problem, scheduling in Grid systems can be viewed as a whole family of problems. This is due to the many parameters that intervene scheduling as well as to the different needs of Grid-enabled applications. In the following, we give some basic concepts of scheduling in Grid systems and identify most common scheduling types. Needless to say, job scheduling in its different forms is computationally hard; it has been shown that the problem of finding optimum scheduling in heterogeneous systems is in general NP-hard [30].

Basic Concepts and Terminology

Although many types of resources can be shared and used in a Computational Grid, normally they are accessed through an application running in the grid. Normally, an application is used to define the piece of work of higher level in the Grid. A typical grid scenario is as follows: an application can generate several jobs, which in turn can be composed of sub-tasks, in order to be solved; the grid system is responsible for sending each sub-task to a resource to be solved. In a simpler grid scenario, it is the user who selects the most adequate machine to execute its sub-tasks. However, in general, grid systems must dispose of schedulers that automatically and efficiently find the most appropriate machines to execute an assembly of tasks.

New characteristics of Scheduling in Grids

The scheduling problem in distributed systems is not new at all; as a matter of fact it is one of the most studied problems in the optimization research community. However, in the grid setting there are several characteristics that make the problem different from its traditional version of conventional distributed systems. Some of these characteristics are the following:

• **The dynamic structure of the Computational Grid**. Unlike traditional distributed systems such as clusters, resources in a Grid system can join or leave the Grid in an unpredictable way. It could be simply due to loosing connection to the system or because their owners switch off the machine or change the operating system, etc. Given that the resources cross different Administrative domains, there is no control over the resources.

• **The high heterogeneity of resources**. Grid systems act as large virtual supercomputers, yet the computational resources could be very disparate, ranging from laptops, desktops, clusters, supercomputers and even small devices of limited computational resources. Current Grid infrastructures are not yet much versatile but heterogeneity is among most important features to take into account in any Grid system.

• **The high heterogeneity of jobs**. Jobs arriving to any Grid system are diverse and heterogeneous in terms of their computational needs. For instance, they could be computing intensive or could be data intensive; some jobs could be 1 Meta-heuristics for Scheduling in Grid systems 7 full applications having a whole range of specifications other could be just atomic tasks. Importantly, Grid system could not be aware of the type of tasks, jobs or applications arriving in the system.

• **The high heterogeneity of interconnection networks.** Grid resources will be connected through Internet using different interconnection networks. Transmission costs will often be very important in the overall Grid performance and hence smart ways to cope with the heterogeneity of interconnection networks is necessary.

• **The existence of local schedulers** in different organizations or resources. Grids are expected to be constructed by the "contribution" of computational resources across institutions, universities, enterprises and individuals. Most of these resources could eventually be running local applications and use their local schedulers, say, a Condor batch system. In such cases, one possible requirement could be to use the local scheduler of the domain rather than an external one.

• **The existence of local policies** on resources. Again, due to the different ownership of the resources, one cannot assume full control over the Grid resources. Companies might have unexpected computational needs and may decide to reduce their contribution to the Grid. Other policies such as rights access, available storage, pay-per-use, etc. are also to be taken into account.

• **Job-resource requirements**. Current Grid schedulers assume full availability and compatibility of resources when scheduling. In real situations, however, many restrictions and/or incompatibilities could be derived from job and resource specifications.

• **Large scale of the Grid system**. Grid systems are expected to be large scale, joining hundreds or thousands of computational nodes world-wide. Moreover, the jobs, tasks or applications submitted to the Grid could be large in number since different independent users and/or applications will send their jobs to the Grid without knowing previous workload of the system. Therefore, the efficient management of resources and planning of jobs will require the use of different types of scheduling (super-schedulers, meta-schedulers, decentralized schedulers, local schedulers, resource brokers, etc.) and their possible hierarchical combinations.

• **Security.**This characteristic, which is in existing in classical scheduling, is an important issue in Grid scheduling. Here the security can be seen as a two-fold objective: on the one hand, a task, job or application could have a security requirement to be allocated in a secure node, that

is, the node will not "watch" or access the processing and data used by the task, job or application. On the other hand, the node could have a security requirement, that is, the task, job or application running in the resource will not "watch" or access other data in the node.

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 as cited in [2,16,36].

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.

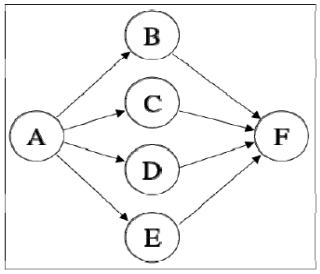


Figure 3-1: Parallel Flow [2]

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.

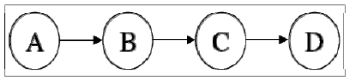


Figure 3-2: Serial Flow [2]

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.

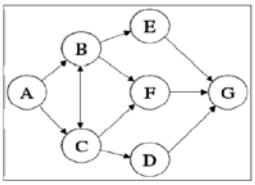
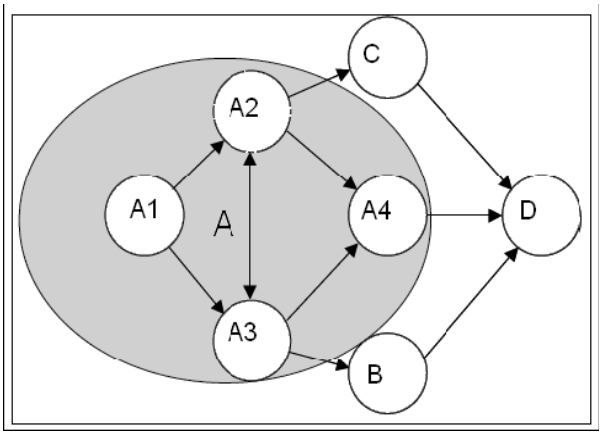


Figure 3-3: Networked Flow [2]

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.



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.

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.

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.

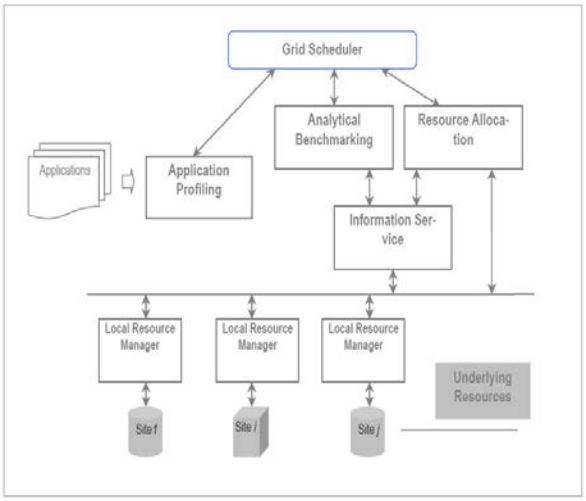
In the following, some common performance goals in both classes are listed:

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.
- **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.
- **Makespan: T**he 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.

A Common Grid Scheduler Architecture

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, Check pointing and Migration: To enable the important feature of rescheduling, three techniques, i.e., job monitoring, check pointing 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.
- **Check pointing** 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. Check pointing is beneficial for enabling us to resume a job instead of re-running it after its long execution.
- **Migration** is the process of transferring currently running jobs to other VM's or nodes in case of VM failures or node failures. Sometimes, migration is possible if

the nodes or VM's are heavily loaded. The main migration policies considered include performance slowdown, target system failure, job cancellation, detection of a better resource, etc.

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

Conclusion and Future Work

In this paper, we have defined the basic terms and summarized the elements of the Grid scheduling problem (GSP).Resources and job properties have been discussed. Various Scheduling algorithms, Types and Characteristics of Grid and Resource management is discussed in this paper.

In the near future, educational institutions and other organizations may construct its local computational Grid and utilize the available CPU power and the idle resources to its maximum. The local Grid should be based on high-speed local network, and enough security need to be provided to the local grid so that outside world should not have direct access to the local resources.

REFERENCES

[1] Maozchen Li and Mark Baker @2005, The Grid :core John Wiley & Sons,Ltd.

[2] Fangpeng Dong and Selim G.AKI Jan 2006,Scheduling Algorithms for Grid computing: state of the Art and Open problems: Technical report no 2006-504,School of Computing, Queen's University Kingston,Ontario.

[3] Jennifer M Schopf,Ten Actions when Scheduling ,the User as a Grid Scheduler, Mathematics and Computer Divison, Argonne National Laboratory.

[4]A Survey on Grid Technologies and Resource Management Systems,chapter2.

[5] Alain Andrieux Globus Alliance ,Dave Berry October 21-22nd 2003 Open Issues in Grid Scheduling, Report of the workshop held at the e-Science Institute,National e-Science Centre,Jon Garibaldi University of Nottingham, Stephen Jarvis, University of Warwick, Jon MacLaren, University of Manchester, Djamila Ouelhadj, University of Nottingham, Dave Snelling, Fujitsu Labs Europe.

[6]Klaus Krauter, Rajkumar Buyya and Muthucumaru Maheswaran, 2001,A taxonomy and survey of grid resource management systems for distributed computing ,John Wiley & Sons, Ltd.

[7] Er.Vijay Dhir, Dr. Rattan K Datta, Dr.Maitreyee Dutta , 2013,Grid Job Scheduling - A Detailed Study, International Journal of Innovative Research in Science, Engineering and Technology,Vol.2,Issue 10,October 2013.

[8] Pavel Fibich and Lud˘ek Matyska and Hana Rudov´a, 2005,Model of Grid Scheduling Problem, American Association for Artificial Intelligence.

[9] Raksha Sharma, Vishnu Kant Soni, Manoj Kumar Mishra, Prachet Bhuyan,2010,A Survey of Job Scheduling and Resource Management in Grid Computing, World Academy of Science, Engineering and Technology,Vol:4 2010-04-22.

[10]R.Venkatesan,J.RajThilak,2012,Perspective Study on task scheduling in computational grid, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)Volume 1, Issue 8, October 2012

[11]Jayoti Bansal et al,2012,A Survey and Taxonomy of scheduling algorithms in Desktop Gridl, IJCSET |March 2012| Vol 2, Issue 3,963-967.

[12] Dilpreet Kaur, Balwinder Singh, Talwandi Sabo, 2012,A Review: Scheduling in Grid Environment, International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 5, November 2012 .

[13]F.Xhafa,A. Abraham (Eds.)2008,Meta-heuristics for Grid Scheduling Problems, Meta. for Sched. in Distri. Comp. Envi., SCI 146, pp. 1–37, 2008. Springer-Verlag Berlin Heidelberg 2008

[14]M. Senobari, Michal Drozdowicz, Maria Ganzha, Marcin Paprzycki, Richard Olejnik, et al. 2009,Resource Management in Grids: Overview and a discussion of a possible approach for an Agent-Based Middleware, Saxe-Coburg Publications, Stirlingshire,UK. PARALLEL, DISTRIBUTED AND GRID COMPUTING FOR ENGINEERING, Stirlingshire,UK, pp 141-164, 2009, Computational Science, Engineering & Technology Series, ISSN 1759-3158.

[15] Nabeel Zanoon, Nashat Al Bdour, Evon ,2014, Abu-Taieh ,Survey of Algorithm: Scheduling Systems and Distributed Resource Management in Grid, International Journal of Computer Applications (0975 – 8887) Volume 98– No.1, July 2014.

[16] Yanmin Zhu, Lionel M. Ni, 2013,A Survey on Grid Scheduling Systems, Technical Report SJTU_CS_TR_200309001, Department of

Computer Science and Engineering, Shanghai JiaoTong University, 2013

[17] Patel Sohil K ,Survey Report of Job Scheduler on Grids, International Journal of Emerging Research in Management &Technology ISSN: 2278-9359 (Volume-2, Issue-4) .

[18] Rajkumar Buyya, David Abramson, and Jonathan Giddy,Grid Resource Management, Scheduling and Computational Economy,2000,In Proceedings of the 2nd International Workshop on Global and Cluster Computing (WGCC'2000) Tsukuba/Tokyo, Japan, March 15 - 17, 2000.

[19] Ye Huanga, Nik Bessis ,Peter Norrington , Pierre Kuonen, Beat Hirsbrunner ,2013,Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm,Future Generation Computer Systems 29 (2013) 402–415.

[20] Ruay-Shiung Chang, Jih-Sheng Chang, Po-Sheng Lin ,An ant algorithm for balanced job scheduling in grids, Department of Computer Science and Information Engineering, National Dong Hwa University, Shoufeng Hualien, 974 Taiwan, ROC., Future Generation Computer Systems 25 (2009) 20–27.

[21] IKatia Leal a,, Eduardo Huedob, Ignacio M. Llorente, 2009, A decentralized model for scheduling independent tasks in Federated Grids,Future Generation Computer Systems 25 (2009) 840-852.

[22] Y.-H. Lee et al(2011) Improving job scheduling algorithms in a grid environment, Future Generation Computer Systems 27 (2011) 991–998.

[23]Quezada-Pina et al,2012,Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids,Future Generation Computer Systems 28 (2012) 965–976.

[24] S. Smanchat et al., 2013,Scheduling parameter sweep workflow in the Grid based on resource competition, Future Generation Computer Systems 29 (2013) 1164–1183.

[25] R.-S. Chang et al, 2012, An Adaptive Scoring Job Scheduling algorithm for grid computing, Information Sciences 207 (2012) 79–89.

[26] X. Tang et al, 2012,A hierarchical reliability-driven scheduling algorithm in grid systems,J.Parallel Distrib. Comput. 72 (2012) 525–535.

[27] *Laizhi Wei et al* , 1975,An Improved Ant Algorithm for Grid Task Scheduling Strategy, *Physics Procedia 24 (2012) 1974 – 1981* 1975.

[28] *Syed Nasir Mehmood Shah et al,2011,*Dynamic Multilevel Hybrid Scheduling Algorithms for Grid Computing, *Procedia Computer Science 4 (2011) 402–411.*

[29] Sara Kardani-Moghaddam, Farzad Khodadadi, Reza Entezari-Maleki, Ali Movaghar,2011,A Hybrid Genetic Algorithm and Variable Neighborhood Search for Task Scheduling Problem in Grid Environment, published by sevier Ltd. Selection and/or peer-review under responsibility of Harbin Universityof Science and Technology.

[30] Syed Nasir Mehmood Shaha, M Nordin B Zakariaa, Nazleeni Harona, AhmadKamil,Bin Mahmooda, Ken Naonob , 2012,Design and Evaluation of Agent Based Prioritized Dynamic Round Robin Scheduling Algorithm on Computational Grids,*AASRI Procedia1(2012) 531 – 543.*

[31] Saeed Parsa and Reza Entezari- Maleki ,2009 RASA: A New Task Scheduling Algorithm in Grid Environment, World Appl. Sci. J., 7 (Special Issue of Computer & IT): 152-160, 2009

[32] Luiz F. Bittencourt, Edmundo R. M. Madeira, Nelson S. da Fonseca

,2012,Scheduling in hybrid clouds, , IEEE Communications Magazine 50(9):42-47 *(*2012*).*

[33]Nikolaos D. Doulamis,Anastasios D. Doulamis, Emmanouel A. Varvarigos, and Theodora A. Varvarigou,2007,Fair Scheduling Algorithms in Grids,IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 18, NO. 11, NOVEMBER 2007.

[34] F. Xhafa, A. Abraham ,2010,Computational models and heuristic methods for Grid scheduling problems, Future Generation Computer Systems 26 (2010) 608621.

[35] Sanjaya Kumar Panda, Pratik Agrawal, Pabitra Mohan Khilar and Durga Prasad Mohapatra,2014 Skewness-Based Min-Min Max-Min Heuristic for Grid Task Scheduling, ACCT'14 proceedings of the 2014th Fourth International Conference on Advanced Computing & Communication Technologies, Pages 282-289,2014, ISBN: 978-1-4799-4910-6.

[36] Michael Walker, 2001,A Framework for Effective Scheduling of Data-Parallel Applications in Grid Systems, May 2001.[37] Gunjan Aggarwal, Meenakshi Kamboj, Charanjit Singh, Preeti Sharma,2012,A Novel Resource Scheduling Algorithm for Computational Grid, ,International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 4– No.3, September 2012.