# 1

# Meta-heuristics for Grid Scheduling Problems

Fatos Xhafa[1] and Ajith Abraham[2]

[1] Departament de Llenguatges i Sistemes Informtics, Universitat Politcnica de Catalunya Barcelona, Spain
fatos@lsi.upc.edu
[2] Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, Norway
ajith.abraham@ieee.org

**Summary.** In this chapter, we review a few important concepts from Grid computing related to scheduling problems and their resolution using heuristic and meta-heuristic approaches. Scheduling problems are at the heart of any Grid-like computational system. Different types of scheduling based on different criteria, such as static vs. dynamic environment, multi-objectivity, adaptivity, etc., are identified. Then, heuristics and meta-heuristics methods for scheduling in Grids are presented. The chapter reveals the complexity of the scheduling problem in Computational Grids when compared to scheduling in classical parallel and distributed systems and shows the usefulness of heuristics and meta-heuristics approaches for the design of efficient Grid schedulers.

**Keywords:** Grid Computing, Scheduling, Independent Scheduling, Grid workflow, Multi-objective Optimization, Heuristics, Meta-heuristics.

## 1.1 Introduction

Grid Computing is a powerful computing paradigm penetrating each time more in every activity of our lives! Grid computing is about benefiting from large computing power never known before, is about scientific progress, business and much more! What would it mean if:

- A researcher from Computer Science could solve to optimality his favorite NP-hard problem within a few hours?
- A researcher from Chemistry could obtain a new drug design not known before?
- A researcher from Biomedicine could discover the DNA sequencing and use it for investigating diseases?
- A climate forecast center could predict in advance a possible tsunami?
- A medical team could remotely run a complex surgery operation using virtual laboratories?
- An economist could analyze almost on real time portfolio values?
- A student of online distance university could contribute his computer to a computational infrastructure and work online with his team for achieving the academic goals?

F. Xhafa, A. Abraham (Eds.): Meta. for Sched. in Distri. Comp. Envi., SCI 146, pp. 1–37, 2008.
springerlink.com 

- An enterprise could never run short of computational resources?
- ...an many more real life scenarios?

It would really mean increasing our knowledge on complex problems, improving our lives, improving our productivity and achieving ambitious goals not possible before! Today all these are possible thanks to advances in Grid Computing!

Grid computing and Grid technologies have primarily emerged for scientific and technical work, where geographically distributed computers, linked through Internet, are used to create virtual supercomputers of vast amount of computing capacity able to solve complex problems from eScience in less time than known before. Thus, within the last years we have witnessed how Grid Computing has helped to achieve breakthroughs in meteorology, physics, medicine and other computing-intensive fields. Examples of such large scale applications are known from optimization (e.g. Casanova et al. [17], Goux et al. [34], Wright [66], Wright et al. [43]), Collaborative/eScience Computing (e.g. Newman et al. [51], Paniagua et al. [54]), Data-Intensive Computing (e.g. Beynon al. [6]), to name a few.

Grid computing is still in the development stage, and most projects are still from academia and large IT enterprises; it has however developed very quickly and more and more scientists are currently engaged to solve many challenges in Grid Computing. Among these, improving its efficiency is imperative! The question is:

> *"How to make use of millions of computers world-wide, ranging from simple laptops, to clusters of computers and supercomputers connected through heterogenous networks in an efficient, secure and reliable manner?"*

The above question is a real challenge for Grid computing community. The good news are the reported advances in both scientific research in Grid Computing and technological achievements and software development for enabling Grid computing systems. Software packages exist and have been successfully deployed and it is now possible to build Grid systems joining together both single computers and clusters of computers yet, *the challenging problem of dynamically and adaptively allocating resources in response to demanding application requests remains unsolved*. For the majority of grid systems, scheduling is a very important mechanism. In the simplest of cases, scheduling of jobs can be done in a blind way by simply assigning the incoming tasks to the compatible resources according to their availability. Nevertheless, it is a lot more profitable to use more advanced and sophisticated schedulers. Moreover, the schedulers would generally be expected to react to the dynamics of the grid system, typically by evaluating the present load of the resources, and notifying when new resources join or drop from the system. Additionally, schedulers can be organized in a hierarchical form or can be distributed in order to deal with the large scale of the grid.

In this chapter, we focus on the design of efficient Grid schedulers using heuristics and meta-heuristics methods. Heuristic and meta-heuristics methods have

proven to be efficient in solving many computationally hard problems. They are showing their usefulness also in the Grid Computing domain, especially for scheduling and resource allocation. We analyze why heuristics and meta-heuristics methods are good alternatives to more traditional scheduling techniques and what make them appropriate for Grid scheduling. An important issue here is how to formally define the Grid scheduling problem. We have presented the most important and useful computational models for this purpose.

The rest of the chapter is organized as follows. We present in Section 1.2 a few important concepts from Grid computing, introduce a few types of Grids in view of needs for different types of scheduling and resource allocation. Then, in Section 1.3 we identify different types of scheduling problems arising in Grid systems. In Section 1.4, we focus in the current state of using heuristic and meta-heuristic methods for solving scheduling problems in Grid systems, as *de facto* approaches for dealing with the complexity of the problem. A few other issues such as security and grid services scheduling are discussed in Section 1.5. We end the chapter in Section 1.6 with a few conclusions.

## 1.2 The *many* Grids

The present state of the computation systems is, in some aspects, analogous to that of the electricity systems at the beginning of the 20th century. At that time, the generation of electricity was possible, but still it was necessary to have available generators of electricity. The true revolution that permitted its establishment was the discovery of new technologies, namely the networks of distribution and broadcast of electricity. These discoveries made possible to provide a reliable, low price service and thus the electricity became universally accessible.

By analogy, the term *grid* is adopted to designate a computational infrastructure of distributed resources, highly heterogeneous (as regards their computing power and architecture), interconnected by heterogeneous communication networks and by a middleware that offers reliable, simple, transparent, efficient and global access to their potential of computation.

The Grid Computing domain has witnesses a fast development over a relatively short time period, pushed by important technology advancements and interest of large IT companies such as IBM, Sun Microsystems, Oracle and HP. The roots of Grid Computing can be traced back to the late 1980s and the first concept that laid the basis of today's Grid systems were developed by researchers from distributed super-computing for numerical or optimization with particular emphasis on scheduling algorithms to achieve high performance computing (e.g. Condor-G). By the late 1990s, the term of Computational Grids and Grid Computing were popularized by Foster et al. [26, 27] who developed the Globus toolkit as a general middleware for Grid Systems. Since then, Grid Computing, Grid systems and Grid technology are advancing in unstoppable way! In the following subsections we briefly review most important types of Grids pushing Grid technology, actually, it is by large impossible to review all existing types of Grids and Grid projects running world-wide!

### 1.2.1 Computational Grids

One of the first questions raised by this emerging technology is its utility or the need of disposing computational grids. On the one hand, the computational proposals have usually shown to have a great success in any field of the human activity. Guided by the increase of the complexity of the real life problems, and prompted by the increment of the capacity of the technology, the human activity (whether scientific, engineering, business, personal, etc.) is highly based on computation. Computers are very often used to model and to simulate complex problems, for diagnoses, plant control, weather forecast, and many other fields of interest. Even so, there exist many problems that challenge or exceed our ability to solve them, typically because they require processing a large quantity of operations or data. In spite of the fact that the capacity of the computers continues improving, the computational resources do not respond to the continuous demand for more computational power.

On the other hand, statistical data show that computers are usually infra-utilized. Most of computers from companies, administration, etc. are most of the time idle or are used for basic tasks that do not require the whole computation power. It is pointed out however by several statistic studies that a considerable amount of money is spent for the acquisition of these resources. One of the main objectives of the grid technology is, therefore, to benefit from the existence of many computation resources through the sharing. As pointed out by Foster & Kesselman *"the sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources..."*

### 1.2.2 Scavenging Grids

In a simple Computational Grid, such as united devices, the politics of "*scavenging*" is applied. This means, each time a machine remains idle, it reports its state to the grid node responsible for the management and planning of the resources. Then, this node usually assigns to the idle machine the next pending task that can be executed in that machine. Scavenging normally hinders the owner of the application, since in the event that the idle machine changes its state to be busy with tasks not coming from the grid system, the application is suspended or delayed. This situation would create completion times not predictable for grid-based applications. With the objective of having a predictable behavior, the resources participating in the grid often are dedicated resources (exclusive use in the grid), and they do not suffer from preemptions caused by external works. Moreover, this permits the tools associated to the schedulers (generally known as *profilers*) to compute the approximate completion time for an assembly of tasks, when their characteristics are known in advance. Sethi@home project is an example of scavenging Grids.

### 1.2.3 eScience Grids

Under the name of eScience Grids are known types of Grids that are primarily devoted to the solution of problems from science and engineering. Such Grids give

support to the computational infra-structure (access to computational and data resources) needed to solve many complex problems arising in areas of science and engineering. Representative examples are UK eScience Grid, German D-Grid, BIG GRID (the Dutch e-Science Grid) and French Grid'5000, to name a few.

### 1.2.4 Data Grids

Data grids are Grid computing systems that primarily deal with data repositories, sharing, access and management of large amounts of distributed data. Many scientific and engineering applications require access to large amounts of distributed data, however, different data could have their own format. An application that needs access to data in different source data needs transparent and secure access to the data. In such Grid systems many types of algorithms, such as replication, are important to increase the performance of Grid enabled applications that use large amount of data. Also, data movement is an issue here in order to achieve high throughput.

### 1.2.5 Enterprise Grids

Although Grid technologies were motivated by High Performance Computing and have been used for several years now in scientific labs, nowadays Grid computing is becoming a significant component of business. Indeed, today's e-business must be able to respond to increasing costumer demands and adjust dynamically and efficiently to marketplace shifts and customer demands. Enterprise Grids make possible to run several projects within one large enterprise or many departments to share resources (computational and/or data) in a transparent way. It should be noted that in such Grids the security and resource policy management issues are not of first concern. Enterprise Grids are thus showing great and innovative changes on how computing is used. Indeed, Grid Computing is envisaged as a significant factor for increasing the productivity and efficiency to the world-wide business. The Grid offers a large potential to solving business problems by facilitating global access to enterprise computing services and data. Examples of enterprise grids are "Sun Grid Engine", "IBM Grid", "Oracle Grid" and "HP Grid".

A new form of enterprise grids is also emerging in institutions, the so called desktop grids, which use the idle cycles of mainly desktop PC's. Small enterprises and institutions usually are equipped with hundreds or thousands of desktops mainly used for office tasks. This amount of PCs is thus a good source for setting up a Grid system for the institution. In this case, the particularity of the Grid system is its unique administrative domain, which makes it easier to manage due to low heterogeneity and volatility of resources (for instance, all PC's could be running under the same OS). Of course, the desktop Grid can cross many administrative domains and in this case the heterogeneity and volatility of the resources is an issue as in a general Grid system setting.

## 1.3 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].

### 1.3.1 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 heterogenous in terms of their computational needs. For instance, they could be computing intensive or could be data intensive; some jobs could be

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 inexisting 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.

*A general definition and terminology*

A precise 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:

**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 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.

**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.

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

**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 co-exists, and they could in general pursue conflicting goals, thus, there is need for interaction between the different schedulers in order to execute the tasks.

**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.

**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.

**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.

*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.

**Resource selection:** Not all resources could be candidates for allocation of task, jobs or applications. Therefore, the selection process is carried out based on job requirements and resource characteristics. The selection process, again, will depend on the scheduling mode. For instance, if tasks were to be allocated in a batch mode, a pool of as many as possible candidate resources will be identified out of the set of all available resources. The selected resources are then used to compute the mapping that meets the optimization criteria.

As part of resource selection, there is also the advanced reservation of resources. Information about future execution of tasks is crucial in this case. Although the queue status could be useful in this case, it is not accurate, especially if priority is one of the task requirements. Another alternative is using prediction methods based on historical data or users specifications of job requirements.

**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).

### 1.3.2 Types of Scheduling in Grids

As mentioned above, 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, we describe the main types of scheduling arising in Grid environments.

#### 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, however, the characteristics of the Grid

environment make the coordination of its execution very complex [15, 76]. As in other types of scheduling, performance is an important issue in order to enable high performance Grid applications. Unlike independent scheduling, it is more difficult to achieve efficient allocation of workflow tasks to the appropriate Grid resources, which largely depends on data movement between tasks and services as well as interaction with different data sources.

Besides the efficiency, Grid workflows should deal with robustness. Certainly, on the one hand, a Grid workflow could run for a long period, which in a dynamic setting increases the possibility of process failure, which could cause failure of the whole workflow if failure mechanisms are not used.

**Centralized, hierarchical and decentralized scheduling**

Both centralized and decentralized scheduling are useful in Grid computing, showing advantages and limitations. Essentially, they differ in the control of the resources as well as knowledge of the overall Grid system. In the case of centralized scheduling, there is more control on resources, the scheduler has knowledge of the system by monitoring of the resources state and therefore, it's easier to obtain efficient schedulers. This type of scheduling, however, suffers from limited scalability. Therefore such type of scheduling are not appropriate for large scale Grids.

Centralized schedulers have a single point of failure. Another way to organize different Grid schedulers is in a hierarchic way, which allows to coordinate scheduler at a certain level. In this case, schedulers at the lowest level in the hierarchy has knowledge of the resources. This scheduler type still suffers from lack of scalability and fault-tolerance, yet it scales better and is more fault-tolerant than centralized schedulers.

In the decentralized or distributed scheduling there is no central entity controlling the resources. The autonomous Grid sites makes it more challenging to obtain efficient schedulers. In decentralized schedulers, the local (site) schedulers play an important role. The scheduling requests, either by local users or other Grid schedulers, are sent to local schedulers, which manage and maintain the state of the queue job. This type of scheduling is more realistic for real Grid systems of large scale although decentralized schedulers could be less efficient than centralized schedulers.

**Static vs. dynamic scheduling**

There are essentially two main aspects that determine the dynamics of the Grid scheduling, namely:

- *The dynamics of job execution*: This refers to the situation when job execution could fail or, in the preemptive mode, job execution is stopped due to the arrival in the system of high priority jobs.
- *The dynamics of resources*: Resources can join or leave the Grid in an unpredictable way, their workload can significantly vary over time, the local policies on usage of resources could change over time, etc.

The above factors decide the behavior of the Grid scheduler, ranging from static to highly dynamic scheduling. For instance, in the static case, there is no job failure and resources are assumed available all the time and fluctuations on computing capacity and workload are not considered. Although this is unrealistic for real Grids, it could be useful to consider for batch mode scheduling: the number of jobs and resources is considered fixed during short intervals of time (time interval between two successive activations of the scheduler) and the computing capacity is also considered unchangeable. Other variations are possible to consider, for instance, just the dynamics of resources but not that of jobs.

### Immediate vs. batch mode scheduling

Immediate and batch scheduling are well-known methods, largely explored for many computing environments and different types of applications. They are also useful for Grid scheduling. In immediate mode, jobs are scheduled as soon as they enter the system, without waiting for the next time interval when the scheduler will get activated or the job arrival rate is small having thus available resources to execute jobs immediately.

In batch mode, tasks jobs or applications are grouped in batches and scheduled as a group. Batch mode scheduling methods are simple and yet powerful heuristics that are distinguished for their efficiency. In contrast to immediate scheduling, batch scheduling could take better advantage of job and resource characteristics in deciding which job to allocate to which resource since they dispose of the time interval between two successive activations of the batch scheduler. Immediate scheduling methods include Opportunistic Load Balancing, Minimum Completion Time, Minimum Execution Time, Switching Algorithm and $k$-Percent Best and among batch mode methods there are Min-Min, Max-Min, Sufferage, Relative Cost and Longest Job to Fastest Resource - Shortest Job to Fastest Resource [1, 9, 44, 67].

### Adaptive Scheduling

The changeability over time of the Grid computing environment requires adaptive scheduling techniques [42] which will take into account both current status of the resources and predictions for their future status with the aim of detecting and avoiding performance deterioration. Rescheduling can also be seen as a form of adaptive scheduling in which running jobs are migrate to more suitable resources.

Casanova *et al.* [18] considered a class of Grid applications with large numbers of independent tasks (Monte Carlo simulations, parameter-space searches, etc.), also known as task farming applications. For these applications with loosely coupled tasks, the authors developed a general adaptive scheduling algorithm. The authors used NetSolve [17] as a testbed for evaluating the proposed algorithm.

Othman *et al.* [52] stress the need for the Grid system's ability to recognize the state of the resources. The authors presented an approach for system adaptation, in which Grid jobs are maintained, using an adaptable Resource Broker.

Huedo *et al.* [38] reported a scheduling algorithm built on top of the GridWay framework, which uses internally adaptive scheduling to reflect the dynamic Grid characteristics.

#### Scheduling in Data Grids

Grid computing environments are making possible applications that work on distributed data and even across different data centers. In such applications, it is not only important to allocate tasks, jobs or application to fastest and reliable nodes but also to minimize data movement and ensure fast access to data. In other terms, data location is important in such type of scheduling. In fact, the usefulness of large computing capacity of the Grid could be compromised by slow data transmission, which could be affected by both network bandwidth and available storage resources. Therefore, data should be "close" to tasks, jobs or applications to achieve efficient access.

### 1.3.3 Computation Models for Formalizing Grid Scheduling

Given the versatility of scheduling in Grid environments, one needs to consider different computation models for Grid scheduling that would allow to formalize, implement and evaluate –either in real Grid or through simulation– different scheduling algorithms. Following we present some important computation models for Grid scheduling. It should be noted that such models have much in common with computation models for scheduling in distributed computing environments. We notice that in all the models described below, tasks, jobs or applications are submitted for completion to a single resource.

#### Expected Time To Compute model

In the model proposed by Ali et al. [5], it is assumed that we dispose of estimation or prediction of the computational load of each task (e.g. in millions of instructions), the computing capacity of each resource (e.g. in millions of instructions per second, MIPS), and an estimation of the prior load of each one of the resources. Moreover, the Expected Time to Compute matrix $ETC$ of size number of tasks by number of machines, where each position $ETC[t][m]$ indicates the expected time to compute task $t$ in resource $m$, is assumed to be known or computable in this model. In the simplest of cases, the entries $ETC[t][m]$ could be computed by dividing the workload of task $t$ by the computing capacity of resource $m$. This formulation is usually feasible, since it is possible to know the computing capacity of resources while the computation need of the tasks (task workload) can be known from specifications provided by the user, from historic data or from predictions [36, 37].

*Modelling heterogeneity and consistency of computing*

The $ETC$ matrix model is able to describe different degrees of heterogeneity in distributed computing environment through consistency of computing. The

consistency of computing refers to the coherence among execution times obtained by a machine with those obtained by the rest of machines for a set of tasks. This feature is particularly interesting for Grid systems whose objective is to join in a single large virtual computer different resources ranging from laptops and PCs to clusters and supercomputers. Thus, three types of consistency of computing environment can be defined using the properties of the ETC matrix: consistent, inconsistent and semi-consistent.

An $ETC$ matrix is said to be consistent if for every pair of machines $m_i$ and $m_j$, if $m_i$ executes a job faster than $m_j$ then $m_i$ executes all the jobs faster than $m_j$. On the contrary, in an inconsistent $ETC$ matrix, a machine $m_i$ may execute some jobs faster than another machine $m_j$ and some jobs slower than the same machine $m_j$. Partially-consistent $ETC$ matrices are inconsistent matrices having a consistent sub-matrix of a predefined size. Further, the $ETC$ matrices are classified according to the degree of job heterogeneity, machine heterogeneity and consistency of computing. Job heterogeneity expresses the degree of variance of execution times for all jobs in a given machine. Machine heterogeneity indicates the variance of the execution times of all machines for a given job.

*Problem instance*

From the above description, it can be seen that formalizing the problem instance is easy under the ETC model; it consists of: a vector of tasks workloads, a vector of computing capacity of machines and the matrix ETC. As we will see in next subsection, it is almost straightforward to define several optimization criteria in this model to measure the quality of a feasible schedule. It is worth noting that incompatibilities among tasks and resources can also be expressed in ETC model, for instance, a value of $+\infty$ to $ETC[t][m]$ would indicate that task $t$ is incompatible with resource $m$. Other restrictions of running a job on a machine can be simulated using penalties to ETC values. It is, however, more complicated to simulate communication and data transmission costs.

### Total Processor Cycle Consumption model

Despite of its interesting properties, the ETC model has an important limitation, namely, the computing capacity of resources remains unchanged during task computation. This limitation becomes more evident when we consider Grid systems in which not only the resources have different computing capacities but also they could change over time due to Grid system's computing overload. The computing speed of resources could be assumed constant only for short or very short periods of time. In order to remedy this, Fujimoto and Hagihara [28] introduced the Total Processor Cycle Consumption (TPCC) model. The total processor cycle consumption is defined as the total number of instructions the Grid resources could complete from the starting time of executing the schedule to the completion time. As in ETC model, task workload is expressed in number of instructions and the computing capacity of resources in number of

instructions computed per unit time. However, now is measured the total consumption of computing power due to Grid application completion. Clearly, this model takes into account that resources could change their computing speed over time, as it happens in large-scale computing systems whose workload is in general unpredictable.

*Problem instance*

A problem instance in TPCC model consists of the vector of task workloads (denoted task lengths in [28]) and a matrix expressing the computing speed of resources. Since the computing speed can change over time, one should fix a short time interval in which the computing speed remains unchanged (for instance, a unit time interval could be considered). Then a matrix $PS$ (for processor speed) is built overtime in which one dimension is processor number and the other dimension is time (discretized by unit time); the component $PS[p][t]$ represents the processor's speed during interval time $[t, t+1)$. As the availability and processing speed of a resource vary over time, the processor speed distribution is used.

This model has shown to be useful for independent and coarse-grain task scheduling, i.e., scheduling in which the computation time in Grid nodes is superior to data transmission time, such as stand-alone applications.

### Grid Information System model

The computation models for Grid scheduling presented so far allow for a precise description of problem instance however they are based on predictions, distributions or simulations. Currently, other Grid scheduling models are developed from a higher level perspective. In the Grid Information System model the Grid scheduler uses task (job or application file descriptions) and resource file descriptions as well as state information of resources (CPU usage, number of running jobs per grid resource), provided by the Grid Information System. The Grid scheduler then computes the best matching of tasks to resources based on the up-to-date workload information of resources. This model is more realistic for Grid environments and is especially suited for the implementation of simple heuristics such as FCFS (First Come First Served), EDF (Earliest Deadline First), SJF (Shortest Job First), etc.

*Problem instance*

The problem instance in this model is constructed, at any point in time, from the information on task file descriptions, resource file descriptions and the current state information on resources.

### Cluster and Multi-Cluster Grids model

Cluster and Multi-Cluster Grids refer to Grid model in which the system is made up of several clusters. For instance the Cluster Grid of an enterprise comprises

different clusters located at different departments of the enterprise. One main objective of cluster grids is to provide a common computing infrastructure at enterprise or department levels in which computing services are distributed to different clusters. More generally, clusters could belong to different enterprises and institutions, that is, are autonomous sites having their local users (both local and grid jobs are run on resources) and usage policies.

The most common scheduling problem in this models is a Grid scheduler which makes use of local schedulers of the clusters. The benefit of cluster grids is to maximize the usage of resources and at the same time, increase throughput for user tasks (jobs or applications). This model has been exploited in Lee and Zomaya [47] for scheduling data-intensive bag-of-tasks applications.

*Problem instance*

The problem instance in this model is constructed, at any point in time, from the information on task file descriptions; again, it is assumed that the workload of each task is known *a priori*. On the other hand, the (multi-)cluster grid can be formally represented as a set of clusters, each one with the information on its resources. Note that in this model the Grid scheduler need not to know the information on resources within a cluster nor the state information or control on every Grid resource. In this way, it is possible to reduce dependencies on Grid information services and respect local policies on resource usage.

### 1.3.4 Grid System Performance and Scheduling Optimization Criteria

Several performance requirements and optimization criteria can be considered for Grid scheduling problem –the problem is multi-objective in its general formulation. We could distinguish proper Grid system performance criteria from scheduling optimization criteria although both performance and optimization objectives allow to establish the overall Grid system performance.

Grid system performance criteria include: CPU utilization of Grid resources, load balancing, system usage, queuing time, throughput, turnaround time, cumulative throughput (i.e. cumulative number of completed tasks) waiting time and response time. In fact other criteria could also be considered for characterizing Grid system's performance such as deadlines, missed deadlines, fairness, user priority, resource failure, etc. Scheduling optimization criteria include: makespan, flowtime, resource utilization, load balancing, matching proximity, turnaround time, total weighted completion time, lateness, weighted number of tardy jobs, weighted response time, etc. Both performance criteria and optimization criteria are desirable for any Grid system; however, their achievement depends also on the considered model (batch system, interactive system, etc.). Importantly, it should be stressed that these criteria are conflicting among them; for instance, minimizing makespan conflicts with resource usage and response time.

Among most popular and extensively studied optimization criterion is the minimization of the makespan. Makespan is an indicator of the general

productivity of the grid system: small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources. Considering makespan as a stand alone criterion not necessarily implies optimization of other objectives. As mention above, its optimization could in fact go in detriment to other optimization criteria. Another important optimization criterion is that of flowtime, which refers to the response time to the user submissions of task executions. Minimizing the value of flowtime means reducing the average response time of the Grid system. Essentially, we want to maximize the productivity (*throughput*) of the grid and at the same time we want to obtain planning of tasks to resources that offer an acceptable *QoS*.

*Makespan, completion time and flowtime*

In Grid scheduling we aim, among other criteria, to minimize the makespan and flowtime. Makespan is the time when finishes the latest task and flowtime is the sum of finalization times of all the tasks. Formally they can defined as:

- minimization of *makespan*: $\min_{S_i \in Sched}\{\max_{j \in Jobs} F_j\}$ and,
- minimization of *flowtime*: $\min_{S_i \in Sched}\{\sum_{j \in Jobs} F_j\}$

where $F_j$ denotes the time when the task $j$ finalizes, $Sched$ is the set of all possible schedules and $Jobs$ the set of all jobs to be scheduled. Note that makespan is not affected by any particular execution order of tasks in a concrete resource, while in order to minimize flowtime of a resource, tasks that have been assigned to should be executed in a ascending order of their workload (computation time).

Completion time of a machine $m$ is the time in which machine $m$ will finalize the processing of the previous assigned tasks as well as of those already planned tasks for the machine. This parameter measures the previous workload of a machine. Notice that this definition requires knowing both the ready time for a machine and the expected time to complete of the tasks assigned to the machine.

The expression of makespan, flowtime and completion time depends on the computational model. For instance, in the ETC model, $completion[m]$ is calculated as follows:

$$completion[m] = ready\_times[m] + \sum_{\{j \in Tasks \ | \ schedule[j]=m\}} ETC[j][m].$$

where $ready\_times[m]$ is the time when machine $m$ will have finished the previously assigned tasks.

Makespan can be expressed in terms of the completion time of a resource, as follows:

$$makespan = \max\{completion[i] \mid i \in Machines\}.$$

Similarly, for the flowtime we use the completion times of machines, but now by first sorting in ascending order according to their $ETC$ values the tasks assigned to a machine. Thus for machine $m$ the flowtime $flowtime[m]$ can be expressed as follows ($S[m]$ is a vector representing the schedule for machine $m$):

```
flowtime[m]=0;
completion = ready_times[m];
for (i = 0; i < S[m].size(); ++i) {
   completion += ETC[S[m][i]][m];
       flowtime[m] += completion;
}
```

In the case of TPCC model, for a schedule $S$ of makespan $M$, the TPCC is expressed as follows:

$$\sum_{p=1}^{m} \sum_{t=0}^{\lfloor M \rfloor -1} S[p][t] + \sum_{p=1}^{m} (M - \lfloor M \rfloor) S[p][\lfloor M \rfloor],$$

where $m$ is the total number of Grid resources used in the schedule, $p$ denotes a processor (resource) and $S[p][t]$ is the speed of processor during time interval $[t, t+1)$. Note that there is no direct relation between TPCC value and makespan value, however the longer makespan, the larger the value of TPCC and vice-versa. In other terms it could be established that any schedule with good TPCC value is a schedule also with good makespan value. In fact it is claimed that the set of makespan optimal schedules is the same as the set of TPCC optimal schedules.

It should be noted that this model is appropriate not only for heuristic-based scheduling methods without guarantee of fitness value of the TPCC but also for *approximation*[1]-based schedulers ensuring a quality of delivered schedule.

*Resource utilization*

Maximizing the resource utilization of the grid system is another important objective. This criterion is gaining importance due to the economic aspects of Grid systems such as the contribution of resources by individuals or institutions in exchange for economic benefits. Achieving a high resource reutilization becomes a challenge in Grid systems given the disparity of computational resources of the Grid. Indeed, to increment the benefit of the resource owners, the scheduler should use any resource, yet this contradicts with the high performance criteria since limited resources could be the bottleneck of the system. It could then be said that from the resource owners perspective, resource utilization is a quality of service criterion.

One possible definition of this parameter is to consider the average utilization of resources. For instance, in the $ETC$ model, for a schedule $S$, it can be defined as follows:

$$avg\_utilization = \frac{\sum_{\{i \in Machines\}} completion[i]}{makespan \times nb\_machines}.$$

and we aim at maximizing this value over all possible schedules.

[1] An approximation algorithm is one that delivers a feasible solution whose fitness value is within a certain bound of the fitness of the optimal solution. Constant factor approximation algorithms, for instance, deliver a solution whose fitness is within a constant factor of the fitness of the optimal solution.

*Matching proximity*

The Grid scheduler should not only map tasks onto resources according to task requirements and resource characteristics but also it aims at matching the tasks to resources that best fit them according to desired computational criteria. Matching proximity is one such facet of the Grid scheduler, which is usually implicitly pursued in Grid schedulers. Expressing this criterion explicitly is sort of more difficult, as compared to other criteria.

In the *ETC* model, matching proximity could be defined as the degree of proximity of a given schedule with regard to the schedule produced by *Minimum Execution Time* (MET) method. MET assigns a job to the machine having the smallest execution time for that job. Observe that a large value of matching proximity means that a large number of jobs is assigned to the machine that executes them faster. Formally, for a schedule $S$, matching proximity can be computed as follows:

$$matching\_proximity = \frac{\sum_{i \in Tasks} ETC[i][S[i]]}{\sum_{i \in Tasks} ETC[i][MET[i]]}.$$

*Turnaround time*

Turnaround time is a useful criterion when the (mean) elapsed time of computation, from the submission of the first task to the completion of the last submitted task, is to be measured. Dominguez et al. [21] considered this objective for scheduling bags-of-tasks applications in desktop Grids. This objective is usually more important in batch scheduling than in interactive applications. Kondo [40] and Kondo et al. [41] characterized four real desktop grid systems and designed scheduling heuristics based on resource prioritization, resource exclusion, and task replication for fast application turnaround.

*Total weighted completion time*

This criterion is appropriate when user's tasks, jobs or applications have priorities. As usually, this criterion is implemented through weights associated to tasks [33, 25] and thus the weighted completion time is expressed as:

$$Total\ weighted\ completion\ time = \sum_{j \in Jobs} w_j F_j$$

where $w_j$ is the priority (weight) of job $j$ and $F_j$ denotes the time when the task $j$ finalizes (completion time of job $j$). As in the case of flowtime, this criterion can be seen as QoS to the Grid user.

In a similar way are defined the *total weighted tardiness* and the *weighted number of tardy jobs* for the case of jobs having due dates.

*Average Weighted Response Time*

In interactive Grid applications, response time is an important parameter. Let $w_j$ be the weight associated to job $j$, $F_j$ its finalization time and $R_j$ its submission time to the Grid system. This criterion can then be expressed as follows:

$$\frac{\sum_{j \in Jobs} w_j (F_j - R_j)}{\sum_{j \in Jobs} w_j}.$$

where $(F_j - R_j)$ is the response time of job $j$. In [24, 60], the response time of a job is weighted by its resource consumption (long jobs have larger resource consumption than short jobs) to balance the impact of short jobs vs. long jobs with a higher resource consumption.

Similarly can be defined the *average weighted wait time*, in which the wait time is defined as the difference between the starting time (when job starts execution) and submission time.

### 1.3.5 Multi-objective Optimization Approaches

As described in the previous subsections, Grid scheduling is multi-objective in its general formulation. Therefore, the optimization criteria, when considered together, have to be combined in a way that a good tradeoff among them is achieved. There are several approaches in multi-objective optimization theory to deal with the multi-criteria condition of the problem. Among them we could distinguish the hierarchical and the simultaneous approach.

*Hierarchical approach*

This approach is useful when we would like, depending on the type of the application or Grid scenario, to establish the priority among the criteria. For instance, in a high performance computing we could give more priority to the makespan and less priority to the response time; yet, if the user requirements are concerned, we could consider the reverse priority. Let $c_i$, $1 \leq i \leq N$ be a set of optimization criteria. In the hierarchic approach, these criteria are sorted by their priority, in a way that if a criterion $c_i$ is of smaller importance than criterion $c_j$, the value for the criterion $c_j$ cannot be varied while optimizing according to $c_i$. These approach has the limitations that one should a priori establish the priority among the criteria and it is not possible to optimize more than one criterion at a time. Nonetheless, its is especially useful when the criteria are measured in different units and can't be combined in a single aggregate objective function (for instance, optimizing makespan and the number of tardy jobs).

This approach has been considered in Xhafa [16, 68, 73] for the independent job scheduling under ETC model.

*Simultaneous approach*

In this approach, an optimal planning is that in which any improvement with respect to a criterion causes a deterioration with respect to another criterion.

Dealing with many conflicting optimization criteria at the same time has certainly a high computation cost. It should be addressed through Pareto[2] optimization theory [23, 62]. However, in the Grid scheduling problem, rather than knowing many Pareto points in solution space, we could be interested to know a schedule having a tradeoff among the considered criteria and which could be computed very fast. Therefore, we can consider a small number of objectives at the same time, which in general suffices for practical applications (usually two or three criteria at the same time would suffice for practical purposes).

In the Pareto optimization theory we could distinguish two different approaches:

**(a) Weighted sum approach:** in this case the optimization criteria are combined in a single aggregate function, which is then solved via heuristic, metaheuristic, AI and hybrid approaches for single-objective problems. There are two issues here: the first is how to combine the different objectives in a meaningful way in a single objective function –in fact this is not always possible! The other problem is that suitable values to the weights of the criteria should be found, which *per se* introduces new variables to the problem definition. For practical cases, however, one could fix a priori the weights either based on a certain (user, application, system performance) priority or conduct a tuning process to identify appropriate values.

**(b) General approach:** In the general approach the objective is to efficiently compute the Pareto optimal front [23, 62]. Many classes of meta-heuristics algorithms have been developed for multi-objective optimization, e.g., Multi-objective Genetic Algorithms (MOGA) [22].

As an example let's consider the case a) when makespan and flowtime are considered simultaneously. As mention before, the first concern is to combine them into a single meaningful objective function. Indeed, when summing them up, we have to take into account that even though makespan and flowtime are measured in the same time unit, the values they can take are in incomparable ranges, due to the fact that flowtime has a higher magnitude order over makespan, and its difference increases as more jobs and machines are considered in the Grid system. In order to deal with this we consider the normalized or mean flowtime: $flowtime/nb\_machines$. Next we have to weight both values to balance their importance:

$$fitness = \lambda \cdot makespan + (1 - \lambda) \cdot mean\_flowtime.$$

In Xhafa et al. [16, 68, 71, 72, 73] the value of $\lambda$ is fixed, based on preliminary tuning, to $\lambda = 0.75$, that is, more priority is given to makespan. In many metaheuristic implementations, it was observed that this single aggregate objective function shows good performance and outperforms known approaches in the literature for the independent Grid scheduling.

[2] Vilfredo Pareto, 1848-1923, Italian economist. He introduced the notion of Pareto-optimality, the idea that a society is enjoying maximum ophelimity *(economic satisfaction)* when no one can be made better off without making someone else worse off.

## 1.4 Heuristics and Meta-heuristics Methods for Scheduling in Grids

From the exposition in the previous sections, it is clear that Grid scheduling problem is really challenging. Dealing with the many constraints and optimization criteria in a dynamic environment is very complex and computationally hard. Meta-heuristic approaches are considered undoubtedly the *de facto* approach. Why meta-heuristics are useful for scheduling in Computational Grids? Following we point out the main reasons that explain the strength of meta-heuristics approaches for designing efficient Grid schedulers:

- *Meta-heuristics are well-understood*: there is a vast body of literature for meta-heuristic approaches. Meta-heuristics have been studied for a large number of optimization problems, from theoretical, practical and experimental perspectives. Certainly, the known studies, results and experiences with meta-heuristic approaches are a good starting point for designing meta-heuristics based Grid schedulers.
- *No "need" for optimal solutions*: In Grid scheduling problem, for most practical applications, any scheduler delivering good quality planning of jobs would suffice rather than searching for optimality. In fact, in highly dynamic Grid environment, there is not possible to even define optimality of planning, as it is defined in combinatorial optimization. This is so due to the fact that Grid schedulers run as long as the Grid system exist and thus the performance is measured not only for particular applications but also in the long run. It is well-known that meta-heuristics are able to compute in short time high quality feasible solutions. Therefore, in such situation meta-heuristics are among best candidates to cope in practice with Grid scheduling.
- *Efficient solutions in short time:* the research work on meta-heuristics has by large tried to find ways to avoid getting stuck in local optima and ensure convergence to sub-optimal or optimal solutions. However, meta-heuristics dispose of mechanisms that allow to "play" with the convergence speed. For instance, in Genetic Algorithms, by choosing appropriate genetic operators one can achieve a very fast convergence of the algorithm to local optima. Similarly, in Tabu Search method, one can work with just short-term memory (recency) in combination with intensification procedure to produce high quality feasible solutions in very short time. This feature of meta-heuristics is very useful for Grid schedulers in which we might want to have a very fast reduction in makespan, flowtime and other parameters.
- *Dealing with multi-objective nature:* Meta-heuristics has proven to efficiently solve not only single objective optimization problems but also multi-objective optimization problems as is the case of Grid scheduling.
- *Appropriateness for periodic and batch scheduling:* Periodic scheduling is a particular case of Grid scheduling. It arises often when companies and users submit their applications to the Grid system periodically. For instance, a bank may wish to run once a month an application that processes the log file keeping bank's clients transaction activity with the bank online system.

In this case suitable resource provisioning can be done in the Grid infrastructures and, which is more important in our context, there are no strong time restrictions. This means that we can run meta-heuristics based schedulers for longer execution times and increase significantly the quality of planning of jobs to resources. Similarly, in batch scheduling, we could run the meta-heuristics based scheduler for the time interval comprised within two successive batches activations.

- *Appropriateness for decentralized approaches:* Since Grid systems are expected to be large or very large scale, decentralization and co-existence of many Grid schedulers in the system is desirable. We could thus have many instances of the meta-heuristics based schedulers running in the system which are coordinated by higher level schedulers.
- *Hybridization with other approaches:* Meta-heuristics can be easily hybridized with other approaches. This is useful to make Grid schedulers to better respond to concrete types of Grid infrastructures, specific types of applications etc. The hybridization has in general shown to produce better solutions than those delivered by single approaches; in fact, meta-heuristics are themselves hybrid approaches.
- *Designing robust Grid schedulers:* The changeability of the Grid environment over time is among the factors that directly influences the performance of the Grid scheduler. A robust scheduler would be one that is able to deliver high quality planning even under constant changes of the characteristics of the Grid infrastructure such as changeability in heterogeneity of resources, of the underlying interconnection networks, in heterogeneity of jobs, etc. Evidence in meta-heuristics literature exist that in general meta-heuristic approaches are robust.
- *Libraries and frameworks for meta-heuristics:* Since meta-heuristic approaches are high level approaches, many libraries and frameworks have been developed in the literature. For instance, Mallba library [1], Paradiseo [14] and EasyLocal++ [45] are such libraries. These libraries can be easily used for Grid scheduling problem; for instance, the meta-heuristic approaches in Xhafa et al. [16, 72, 73] use skeletons defined in Mallba Library. It is worth to note that libraries have been also developed for the meta-heuristics to deal with multi-objective optimization case.

In the next subsections we briefly review most important heuristic and meta-heuristic approaches and the benefits of using them for Grid scheduling problem (the reader is referred to [31, 48] for a survey on meta-heuristic approaches).

### 1.4.1 Local Search Based Heuristic Approaches

Local search heuristics [39] is a family of methods that explore the solution space by jumping from one solution to another one and constructing thus a path in solution space with the aim of finding the best solution for the problem. Methods in this family range from simple ones such as Hill Climbing, Simulated Annealing to more sophisticated ones such as Tabu Search method.

Simple local search methods (Hill Climbing-like) are of interest, at least, for two reasons: (1) they produce a feasible solution of certain quality within very short time; and, (2) they can be used to feed (initialize) population-based meta-heuristics with genetically diverse individuals. Such methods has been studied for the scheduling under ETC model in Ritchie and Levine [58]. Xhafa [68] used several local search methods in implementing Memetic Algorithms for the same problem.

Simulated Annealing (SA) is more powerful than simple local search by accepting also worse solutions with certain probability. This method has been proposed for Grid scheduling problem by Abraham et al. [1] and Yarkhan and Dongarra [75].

Tabu Search [32] is more sophisticated and usually requires more computation time for computing good solutions. However, its mechanisms of tabu lists, aspiration criteria, intensification and diversification make it very powerful search algorithm. Abraham et al. [1] also considered Tabu Search as candidate solution method for the problem. Ritchie [57] implemented the TS for the problem under ETC model and used it in combination with ACO approach. Recently, Xhafa et al. [74] has presented the design, implementation and evaluation of a full TS for the scheduling problem under ETC model. The proposed TS approach showed to outperform Ritchie's approach for the problem.

Following we present the design of simple local search methods for Grid scheduling problem and present some computational results for the problem under ETC model. Makespan and flowtime objectives are considered for this purpose.

*Design of local search methods for Grid scheduling*

Local search methods like Hill Climbing can be applied straightway to the Grid scheduling problem. However, many variations of these methods can be designed by considering different neighborhood structures and move types in order to increase their performance. Indeed, many Hill Climbing versions are obtained by defining appropriate neighborhood relationships. Moreover, different variations are due to the order and the way in which neighboring solutions are visited. For instance, if in each iteration the best neighboring solution is accepted, we have the *steepest descent* version (in minimization case) and *steepest ascent*, in maximization case.

- *Move-based local search*: In this group of methods, the neighborhood is fixed by moving a task from one resource to another one. Thus, two solutions are neighbors if they only differ in a position of their vector of assignments task-resource. The following methods are obtained: (a) *Local Move (LM)*: moves a randomly chosen task from the resource where it was assigned to, to another randomly chosen resource; (b) *Steepest Local Move (SLM)*: moves a randomly chosen task to the resource yielding the largest improvement; (c) *Local MCT Move (LMCTM)*: this method is based on the MCT (*Minimum Completion Time*) heuristic. Here, a task is moved to the resource yielding

the smallest completion time among all the resources; (d) *Local Minimum Flowtime Move (LMFTM)*: applies the movement of a randomly chosen task that yields the largest reduction in the flowtime.

- *Swap-based local search*: In this group of methods, the neighborhood is fixed by swapping two tasks of different resources. This group includes: (a) *Local Swap (LS)*: the resources of two randomly chosen tasks are swapped; (b) *Steepest Local Swap (SLS)*: the movement swap yielding the largest improvement is is applied; (c) *Local MCT Swap (LMCTS)*: in this case, a randomly chosen task $t_1$ is swapped with a task $t_2$ so that the maximum completion_time of the two implied resources is the smallest of all possible exchanges; (d) *Local MFT Swap (LMFTS)*: the exchange of the two tasks yields the largest reduction in the value of flowtime; and, (e) *Local Short Hop (LSH)*: this method is based on the the process of *Short Hop* [7]. In our case, pairs of resources are chosen one from the subset of the most loaded resources and the other from the subset of the less loaded resources together with the subset of tasks that are assigned to these resources. In each iteration (*hop*) the swap of a task of a most loaded resource with a task assigned to a less loaded resource is evaluated and accepted if the completion_time of the implied resources is reduced.
- *Rebalance-based local search*: load balancing of resources is used as a criterion for the neighborhood definition. We can thus design: (a) *Local Rebalance (LR)*: the movement from a solution to a neighboring one is done by rebalancing the most loaded resources; (b) *Deep Local Rebalance (DLR)*: applies a movement with the largest improvement in rebalancing; (d) *Local Flowtime Rebalance (LFR)*: the swap is done for a task from the most loaded resource and a task of a resource that reduces the value of the flowtime contributed by the most loaded resource; (e) *Emptiest Resource Rebalance (ERR)*: in this method the aim is to balance the workload of the resources but now the less loaded resource are used as a basis; and, (f) *Emptiest Resource Flowtime Rebalance (ERFR)*: this is similar to the previous method but now the less loaded resource is considered the one that contributes the smallest flowtime.
- *Variable Neighborhood Search (VNS)*: in this method a generalized concept of neighborhood is considered. More precisely, the neighborhood relationship is defined so that two solutions are considered neighbors if they differ in $k$ positions of their vectors of assignments task-resource, where $k$ is a parameter. This method in general could yield better solutions, however its computational cost is higher since the size of the neighborhood is much larger than in the case of simple neighborhood (for $k = 1$, VNS is just the Local Move).

*Computational results for local search methods (ETC model)*

We exemplify the usefulness of the local search methods presented above through their implementation for the independent Grid scheduling under ETC model (see subsection 1.3.3).

For the purposes of this experimental study problem instances from ETC model consisting of 512 jobs and 16 resources are used. The aim was to study

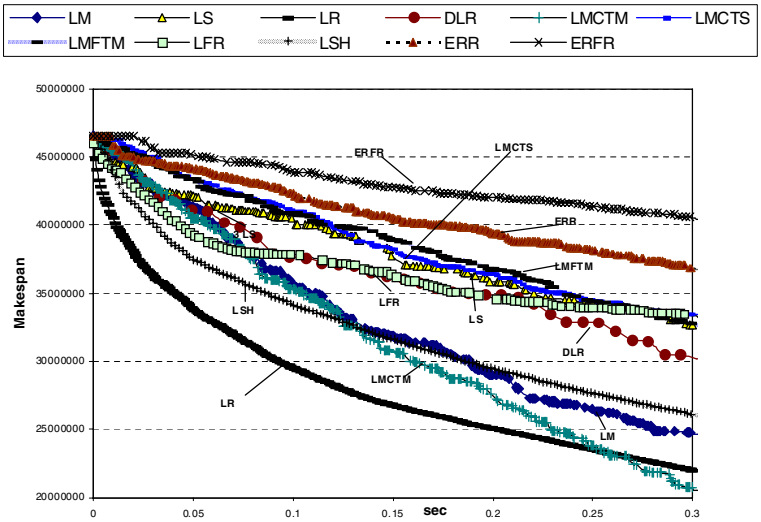


**Fig. 1.1.** Comparison of makespan (in arbitrary time units) reductions obtained with different local search procedures

the makespan reduction of different local search methods presented above (the initial solution –staring point– of the local search was generated randomly). Since local search methods are based on random decisions, 20 independent runs (of 500 iterations each) were performed on the same instance and the performance evaluation is done based on averaged makespan values. We show in Figs 1.1 the graphical representation of the makespan reduction for 11 of the local search methods introduced above; the rest of them (LMFTS, SLS, SLM, ERFR) performed worse and are omitted from the graph.

From Figs 1.1, we can see that: (a) all the considered local search methods achieve a reduction of makespan in very short time; (b) the fastest reduction in makespan is achieved by LR (Local Rebalance) although in the long run LMCTM (Local MCT Move) obtained better makespan reduction; (c) the method ERFR (Emptiest Resource Flowtime Rebalance) based on flowtime reduction performed poorly, which is expected since it tries to minimize flowtime, not the makespan.

On the other hand we measured the makespan reduction of the VNS method for $k = 3$ and $k = 8$. We show in Figs 1.2 the graphical representation of the makespan reduction; we have also included in the graph the LR and LMCTM methods for ease of comparison between VNS and simple local search methods presented above.

From Fig. 1.2 we can see that VNS(18) achieves the fastest reduction of the maksepan but is soon "stagnated" and VNS(3) performs better. This could be explained by the fact that doing a considerable number of movements (8 movements in this case as compared to just three movements in VNS(3)) could

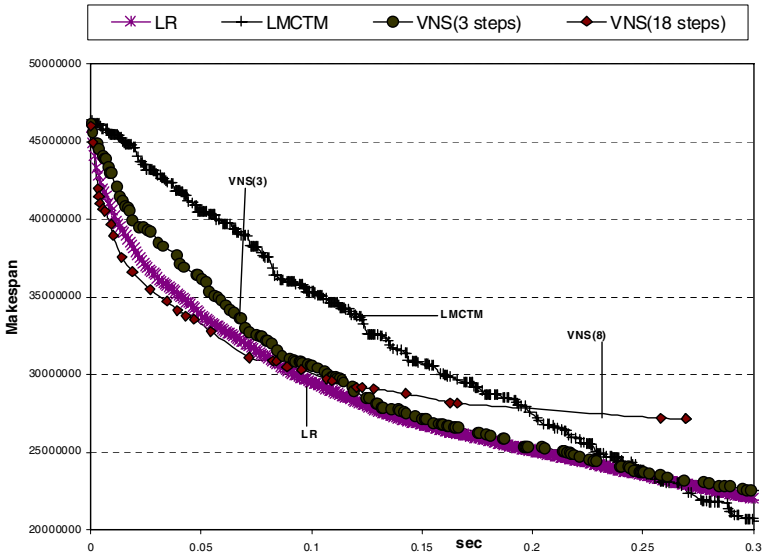


**Fig. 1.2.** Comparison of makespan (in arbitrary time units) reductions obtained with VNS($k$) method for $k = 3$ and $k = 8$

damage the structure of the schedule. It is therefore suggestive to keep the value of $k$ small. It should also be noted that VNS, despite of being considered more powerful method than simple local search, does not perform significantly better than; in fact, LMCTM seems to perform better than VNS in the long run.

### 1.4.2 Population-Based Heuristic Approaches

Population-based heuristics is a large family of methods that has shown their efficiency for solving combinatorial optimization problems. Population based methods usually require large running times if sub-optimal or optimal solutions are to be found. However, when the objective is to find feasible solutions of good quality in short execution times, as in case of Grid scheduling, we can exploit the inherent mechanisms of these methods to increase the *convergence* rapidity of the method.

We could distinguish three different categories of population-based methods: Evolutionary Algorithms (Genetic Algorithms (GAs), Memetic Algorithms (MAs) and their variations), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).

GAs for Grid scheduling problems have been addressed by Abraham et al. [1], Braun et al. [9], Zomaya and Teh [81], Martino and Mililotti [46], Page and Naughton [53], Carretero and Xhafa [16], Gao et al. [29], Xhafa et al. [70, 73].

MAs [49] is a relatively new class of population-based methods, which combine the concepts of evolutionary search and local search by taking advantage of good characteristics of both of them. In this sense MAs could be considered as hybrid

evolutionary algorithms, in fact, MAs arose as an attempt to combine concepts and strategies of different meta-heuristics. There has been few work on MAs for Grid scheduling problem. Xhafa [68] applied unstructured MAs and Xhafa et al. [71] proposed Cellular MAs (structured MAs) for the independent scheduling problem under ETC model.

An ACO implementation for the problem under ETC model has been reported by Ritchie [57]. Abraham et al. [3] proposed an approach for scheduling jobs on Computational Grids using fuzzy PSO algorithm.

*Specific methods for population initialization*

In population-based methods, its is important to dispose a wide variety of initialization methods for the generation the first population. Typically, the initial solutions are generated randomly, however, introducing a few genetically good individuals would be helpful to accelerate the search. Thus, besides a random method, other specific or *ad hoc* methods can be used to generate solutions, among them, the *ad hoc* heuristics Opportunistic Load Balancing (OLB), Minimum Completion Time (MCT), Minimum Execution Time (MET), Switching Algorithm (Switch), K-percent Best (KPB), Min-min, Max-min, Sufferage, Relative-cost and Longest Job to Fastest Resource-Shortest Job to Fastest Resource (LJFR-SJFR) [1, 9, 44, 67].

In [73], the LJFR-SJFR method was used for generating one individual (the rest were generated through random perturbations, that is, by reassignment of a subset of tasks). By monitoring some of the runs, we observed that our GA spends roughly 55-70% of the total number of iterations to reach a solution of the quality of Min-Min method, which is due to the fact that Min-Min performs much better than LJFR-SJFR.

### 1.4.3 Hybrid Heuristics Approaches

Meta-heuristic methods are *per se* hybridized approaches. For instance, MAs combine evolutionary search with local search. However, hybridization among different meta-heuristics has shown to be effective for many problems by outperforming single methods [63]. However, hybrid meta-heuristics have been less explored for the problem. Abraham et al. [1] addressed the hybridization of GA, SA and TS heuristics; the hybridization GA+SA is expected to have a better convergence than pure GA search and GA+TS could improve the efficiency of GA. In these hybridizations a heuristic capable to deal with a population of solutions, such as GA, is combined with two other local search heuristics, such as TS and SA, that deal with only one solution at a time. Another hybrid approach for the problem is due to Ritchie and Levine [57, 59] who combine an ACO algorithm with a TS algorithm for the problem. In [68], a basic unstructured MA is combined with 16 local search algorithms in order to identify the best performance of the resulting MA.

### 1.4.4 Other Approaches

Many other approaches can be applied to Grid scheduling problem. We briefly present them next.

*Hyper-heuristic approaches*

Hyper-heuristic approaches [9] are methods that guide the search, at a higher level as compared to the meta-heuristics approaches, through other heuristic methods for the resolution of optimization problems. Hyper-heuristics have shown effective for scheduling and timetabling (Burke et al. [11]). Hyper-heuristics can also be combined to design hybrid approaches for general scheduling and timetabling problems [8, 10]. They are therefore candidate approaches also for Grid scheduling problem.

Xhafa [69] presented a simple hyper heuristic for the problem, which uses as underlying heuristics a set of *ad hoc* (immediate and batch mode) scheduling methods to provide the scheduling of jobs to Grid resources according to the Grid and job characteristics.

The hyper-heuristic is a high level algorithm, which examines the state and characteristics of the Grid system (jobs and resources), and selects and applies the *ad hoc* method that yields the best planning of jobs. The resulting hyper-heuristic based scheduler can be thus used to develop network-aware applications that need efficient planning of jobs to resources.

*Reinforced learning*

Some research work in the literature addressed the use of reinforced learning techniques for scheduling in Grid systems. Perez *et al.* [55], proposed to implement a Reinforcement Learning based scheduling approach for large Grid computing systems. Vengerov [64] presented a utility-based framework for making repeated scheduling decisions dynamically; the observed information about unscheduled jobs and system's resources is used for this purpose.

*Fuzzy logic, neural networks and QoS approaches*

Zhou et al. [80] used Fuzzy Logic techniques to design an adaptive Fuzzy Logic scheduler, which utilizes the Fuzzy Logic control technology to select the most suitable computing node in the Grid environment. A Fuzzy Neural Networks was proposed by Yu et al. [77] to develop a high performance scheduling algorithm. The algorithms uses Fuzzy Logic techniques to evaluate the Grid system load information, and adopt the Neural Networks to automatically tune the membership functions. Hao et al. [35] presented a Grid resource selection based on Neural Networks aiming at offering QoS on distributed, heterogeneous resources. To this end, the authors propose to select Grid resources constrained by QoS criteria. The resource selection problem is solved using a novel neural networks.

Chunlin and Layuan [20] proposed a joint QoS optimization approach to optimize global QoS by adopting cross-layer design and information exchange among multiple Grid layers.

*Economy-based scheduling*

Economy-based models are important for the design of resource management architecture for Grid systems. Several recent works [4,12,13,19,78] are addressing the resource allocation through market-oriented approaches. These approaches are suitable, on the one hand, to exploit the interaction of different scheduling layers, and on the other, different negotiation and agreement strategies can be implemented for resource allocation.

*Grid services scheduling*

W3C defined a service is a set of actions that form a coherent whole from the point of view of service providers and service requesters. Although this definition originated for web systems, services were defined similarly for Grid systems. There are two aspectes related to Grid scheduling and Grid services: (a) Grid services need to be discovered and scheduled to appropriate resources; for instance, scheduling a service in the Grid system to process a requested transaction; and (b) achieving Grid scheduling functionalities through services. Several recent research work [50, 65, 79] explore these aspects, yet there is still few research work in this direction.

## 1.5 Further Issues

Besides the many aspects and facets of Grid scheduling problem presented in the previous sections, there still remain other issues to be considered. We briefly mention here the Grid security as an important aspect to be considered in Grid scheduling. The security can be seen as a two-fold objective: on the one hand, a task, a 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.

It should be noted that current security approaches are treated at different levels of Grid systems and independently of the Grid schedulers. It is challenging to incorporate the security/trust level as one of the objectives of the scheduling by using trust values that span from very trustworthy to very untrustworthy scale. Moreover, one of the aims to pursue here is to reduce the possible overhead to the Grid scheduler and to the overall system that would introduce a secure scheduling approach.

## 1.6 Conclusions

In this Chapter, we have reviewed the most important concepts from Grid computing related to scheduling problems and their resolution using heuristic and meta-heuristic approaches. After introducing a few important Grid types that

have appeared in the Grid computing domain, we identify different types of scheduling based on different criteria, such as static vs. dynamic environment, multi-objectivity, adaptivity, etc. Our exposition aims to reveal the complexity of the scheduling problem in Computational Grids when compared to scheduling in classical parallel and distributed systems and shows the usefulness of heuristics and meta-heuristics approaches for the design of efficient Grid schedulers. We have reasoned about the importance and usefulness of meta-heuristic approaches for the design of efficient Grid schedulers when considering the scheduling as a multi-objective optimization problem. Also, a few other approaches and current research issues in the context of Grid scheduling are discussed.

## Acknowledgment

The first author acknowledges partial support by Projects ASCE TIN2005-09198-C02-02, FP6-2004-ISO-FETPI (AEOLUS) and MEC TIN2005-25859-E and FORMALISM TIN2007-66523.

## References

1. Alba, E., Almeida, F., Blesa, M., Cotta, C., Díaz, M., Dorta, I., Gabarró, J., León, C., Luque, G., Petit, J., Rodríguez, C., Rojas, A., Xhafa, F.: Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. Parallel Computing 32(5-6), 415–440 (2006)
2. Abraham, A., Buyya, R., Nath, B.: Nature's heuristics for scheduling jobs on computational grids. In: The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), India (2000)
3. Abraham, A., Liu, H., Zhang, W., Chang, T.: Scheduling jobs on computational grids using fuzzy particle swarm algorithm. In: 10th Int. Conf. on Knowledge-Based & Intelligent Information & Engineering Systems. LNCS. Springer, Heidelberg (2006)
4. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the Nimrod-G resource broker. Future Generation Computer Systems Journal 18(8), 1061–1074 (2002)
5. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D.: Task execution time modeling for heterogeneous computing systems. In: Proceedings of Heterogeneous Computing Workshop (HCW 2000), pp. 185–199 (2000)
6. Beynon, M.D., Sussman, A., Catalyurek, U., Kure, T., Saltz, J.: Optimization for data intensive grid applications. In: Third Annual International Workshop on Active Middleware Services, California, pp. 97–106 (2001)
7. Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. of Parallel and Distributed Comp. 61(6), 810–837 (2001)
8. Burke, E., Kendall, G., Landa Silva, D., O'Brien, R., Soubeiga, E.: An ant algorithm hyperheuristic for the project presentation scheduling problem. The 2005 IEEE Congress on Evolutionary Computation 3, 2263–2270 (2005)

9. Burke, E.K., Kendall, G., Newall, J., Hart, E., Ross, P., Schulemburg, S.: Hyper-heuristics: an Emerging Direction in Modern Search Technology. In: Glover, F.W., Kochenberger, G.A. (eds.) Handbook of Meta-heuristics. Kluwer, Dordrecht (2003)
10. Burke, E.K., Kendall, G., Soubeiga, E.: A Tabu-Search Hyperheuristic for Timetabling and Rostering. J. Heuristics 9(6), 451–470 (2003)
11. Burke, E., Soubeiga, E.: Scheduling Nurses Using a Tabu-Search Hyperheuristic. In: Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003), Nottingham, UK, pp. 180–197 (2003)
12. Buyya, R.: Economic-based Distributed Resource Management and Scheduling for Grid Computing. PhD thesis, Monash University, Australia (2002)
13. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In: The 4th Int. Conf. on High Performance Comp., Asia-Pacific, China (2000)
14. Cahon, S., Melab, N., Talbi, E.: ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Meta-heuristics. Journal of Heuristics 10(3), 357–380 (2004)
15. Cao, J., Jarvis, S.A., Saini, S., Nudd, G.R.: GridFlow: Workflow Management for Grid Computing. In: Proc. of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan, May 2003, pp. 198–205 (2003)
16. Carretero, J., Xhafa, F.: Using Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications. Journal of Technological and Economic Development –A Research Journal of Vilnius Gediminas Technical University 12(1), 11–17 (2006)
17. Casanova, H., Dongarra, J.: Netsolve: Network enabled solvers. IEEE Computational Science and Engineering 5(3), 57–67 (1998)
18. Casanova, H., Kim, M., Plank, J.S., Dongarra, J.J.: Adaptive Scheduling for Task Farming with Grid Middleware. Int. J. High Perform. Comput. Appl. 13(3), 231–240 (1999)
19. Chin, S., Lee, J., Yoon, T., Yu, H.: List Scheduling Method for Service Oriented Grid Applications. In: Proceedings of the Second international Conference on Semantics, Knowledge, and Grid, p. 44. IEEE Computer Society, Los Alamitos (2006)
20. Chunlin, L., Layuan, L.: Joint QoS optimization for layered computational grid. Inf. Sci. 177(15), 3038–3059 (2007)
21. Domingues, P., Andrzejak, A., Silva, L.: Scheduling for fast touraround time on institutional desktop grid. CoreGRID TechRep No. 0027
22. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
23. Ehrgott, M., Gandibleux, X.: Approximative solution methods for multiobjective combinatorial optimization. TOP –Trabajos de Investigación Operativa 12(1), 1–88 (2004)
24. Ernemann, C., Hamscher, V., Yahyapour, R.: Benefits of Global Grid Computing for Job Scheduling. In: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing. International Conference on Grid Computing, pp. 374–379. IEEE Computer Society, Washington (2004)
25. Fibich, P., Matyska, L., Rudová, H.: Model of Grid Scheduling Problem. In: Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, pp. 17–24. AAAI Press, Menlo Park (2005)
26. Foster, I., Kesselman, C.: The Grid - Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1998)
27. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid. International Journal of Supercomputer Applications 15(3) (2001)

28. Fujimoto, N., Hagihara, K.: Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid. In: Second International Symposium on Parallel and Distributed Computing (ISPDC 2003), pp. 80–87 (2003)
29. Gao, Y., Rong, H., Huang, J.Z.: Adaptive Grid job scheduling with genetic algorithms. Future Gener. Comput. Syst. 21(1), 151–161 (2005)
30. Garey, M.R., Johnson, D.S.: Computers and Intractability – A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., New York (1979)
31. Gendreau, M., Potvin, J.-Y.: Meta-heuristics in Combinatorial Optimization. Annals of Operations Research 140(1), 189–213 (2005)
32. Glover, F.: Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Op. Res. 5, 533–549 (1986)
33. Gomoluch, J., Schroeder, M.: Market-based Resource Allocation for Grid Computing: A Model and Simulation. In: Middleware Workshops 2003, pp. 211–218 (2003)
34. Goux, J.P., Kulkarni, S., Linderoth, J., Yoder, M.: An enabling framework for master-worker applications on the computational grid. In: 9th IEEE Int. Symposium on High Performance Distributed Computing (HPDC 2000) (2000)
35. Hao, X., Dai, Y., Zhang, B., Chen, T., Yang, L.: QoS-Driven Grid Resource Selection Based on Novel Neural Networks. In: Chung, Y.-C., Moreira, J.E. (eds.) GPC 2006. LNCS, vol. 3947, pp. 456–465. Springer, Heidelberg (2006)
36. Hotovy, S.: Workload evolution on the Cornell Theory Center IBM SP2. In: Job Scheduling Strategies for Parallel Proc. Workshop, IPPS 1996, pp. 27–40 (1996)
37. The Hebrew University Parallel Systems Lab. Parallel workload archive, `http://www.cs.huji.ac.il/labs/parallel/workload/`
38. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. In: 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004), p. 28 (2004)
39. Hoos, H.H., Stützle, Th.: Stochastic Local Search: Foundations and Applications. Elsevier/Morgan Kaufmann (2005)
40. Kondo, D.: Scheduling Task Parallel Applications for Rapid Turnaround on Desktop Grids. Doctoral Thesis, University of California at San Diego (2005)
41. Kondo, D., Chien, A., Casanova, H.: Scheduling Task Parallel Applications for Rapid Turnaround on Enterprise Desktop Grids. Journal of Grid Computing 5(4), 379–405 (2007)
42. Lee, L., Liang, C., Chang, H.: An Adaptive Task Scheduling System for Grid Computing. In: Proceedings of the Sixth IEEE international Conference on Computer and information Technology (CIT 2006), September 20-22, p. 57. IEEE Computer Society, Washington (2006)
43. Linderoth, L., Wright, S.J.: Decomposition algorithms for stochastic programming on a computational grid. Computational Optimization and Applications (Special issue on Stochastic Programming) 24, 207–250 (2003)
44. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing 59(2), 107–131 (1999)
45. Di Gaspero, L., Schaerf, A.: EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In: 4th Metaheuristics International Conference (MIC 2001), pp. 287–292 (2001)
46. Di Martino, V., Mililotti, M.: Sub optimal scheduling in a grid using genetic algorithms. Parallel Computing 30, 553–565 (2004)

47. Lee, Y.C., Zomaya, A.Y.: Practical Scheduling of Bag-of-Tasks Applications on Grids with Dynamic Resilience. IEEE Transactions on Computers 56(6), 815–825 (2007)
48. Michalewicz, Z., Fogel, D.B.: How to solve it: modern heuristics. Springer, Heidelberg (2000)
49. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report No. 826, California Institute of Technology, USA (1989)
50. MacLaren, J., Sakellariou, R., Krishnakumar, K.T., Garibaldi, J., Ouelhadj, D.: Towards Service Level Agreement Based Scheduling on the Grid. In: Workshop on Planning and Scheduling for Web and Grid Services (held in conjunction with the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)), Canada (2004)
51. Newman, H.B., Ellisman, M.H., Orcutt, J.A.: Data-intensive e-Science frontier research. Communications of ACM 46(11), 68–77 (2003)
52. Othman, A., Dew, P., Djemame, K., Gourlay, K.: Adaptive Grid Resource Brokering. In: IEEE International Conference on Cluster Computing (CLUSTER 2003), p. 172 (2003)
53. Page, J., Naughton, J.: Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. AI Review 24, 415–429 (2005)
54. Paniagua, C., Xhafa, F., Caballé, S., Daradoumis, T.: A parallel grid-based implementation for real time processing of event log data in collaborative applications. In: Parallel and Distributed Processing Techniques (PDPT 2005), Las Vegas, USA, pp. 1177–1183 (2005)
55. Perez, J., Kégl, B., Germain-Renaud, C.: Reinforcement learning for utility-based Grid scheduling. In: NIPS 2007 (Twenty-First Annual Conference on Neural Information Processing Systems) Workshops, Vancouver, Canada (2007)
56. Raman, R., Solomon, M., Livny, M., Roy, A.: The classads language. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management: State of the Art and Future Trends, pp. 255–270. Kluwer Academic Publishers, Norwell
57. Ritchie, G.: Static multi-processor scheduling with ant colony optimisation & local search. Master's thesis, School of Informatics, Univ. of Edinburgh (2003)
58. Ritchie, G., Levine, J.: A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. Technical report, Centre for Intelligent Systems and their Applications, University of Edinburgh (2003)
59. Ritchie, G., Levine, J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: 23rd Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2004) (2004)
60. Schwiegelshohn, U., Yahyapour, R.: Analysis of First-Come-First- Serve Parallel Job Scheduling. In: Proceedings of the 9th SIAM Symposium on Discrete Algorithms, January 1998, pp. 629–638 (1998)
61. Schopf, J.M.: Ten Actions when Grid Scheduling. In: Nabrzyski, Schopf, Weglarz (eds.) Grid Resource Management, ch. 2. Kluwer, Dordrecht (2004)
62. Steuer, R.E.: Multiple Criteria Optimization: Theory, Computation and Application. Series in Probability and Mathematical Statistics. Wiley, Chichester (1987)
63. Talbi, E.G.: A Taxonomy of Hybrid Meta-heuristics. J. Heuristics 8(5), 541–564 (2002)
64. Vengerov, D.: Adaptive Utility-Based Scheduling in Resource-Constrained Systems. In: Zhang, S., Jarvis, R. (eds.) AI 2005. LNCS (LNAI), vol. 3809, pp. 477–488. Springer, Heidelberg (2005)

65. Venugopal, S., Buyya, R., Winton, L.: A Grid service broker for scheduling e-Science applications on global data Grids. Concurrency and Computation: Practice and Experience 18(6), 685–699 (2006)
66. Wright, S.J.: Solving optimization problems on computational grids. Optima 65 (2001)
67. Wu, M.Y., Shu, W.: A high-performance mapping algorithm for heterogeneous computing systems. In: Proceedings of the 15th International Parallel & Distributed Processing Symposium, p. 74 (2001)
68. Xhafa, F.: A Hybrid Evolutionary Heuristic for Job Scheduling in Computational Grids, ch. 10. Studies in Computational Intelligence, vol. 75. Springer, Heidelberg (2007)
69. Xhafa, F.: A Hyper-heuristic for Adaptive Scheduling in Computational Grids. International Journal on Neural and Mass-Parallel Computing and Information Systems 17(6), 639–656 (2007)
70. Xhafa, F., Duran, B., Abraham, A., Dahal, K.P.: Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids. In: IEEE CelGrid Workshop, OOstrava, The Czech Republic, June 26-June 28 (to appear, 2008)
71. Xhafa, F., Alba, E., Dorronsoro, B., Duran, B.: Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms. Journal of Mathematical Modelling and Algorithms (accepted, 2008) Published Online DOI: http://dx.doi.org/10.1007/s10852-008-9076-y
72. Xhafa, F., Barolli, L., Durresi, A.: An Experimental Study On Genetic Algorithms for Resource Allocation On Grid Systems. Journal of Interconnection Networks 8(4), 427–443 (2007)
73. Xhafa, F., Carretero, J., Abraham, A.: Genetic Algorithm Based Schedulers for Grid Computing Systems. International Journal of Innovative Computing, Information and Control 3(5), 1–19 (2007)
74. Xhafa, F., Carretero, J., Alba, E., Dorronsoro, B.: Design and Evaluation of a Tabu Search Method for Job Scheduling in Distributed Environments. In: The 11th International Workshop on Nature Inspired Distributed Computing (NIDISC 2008) held in conjunction with the 22th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS 2008), Miami, Florida, USA, April 14-18 (2008)
75. YarKhan, A., Dongarra, J.: Experiments with scheduling using simulated annealing in a grid environment. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 232–242. Springer, Heidelberg (2002)
76. Yu, J., Buyya, R.: A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing 3(3), 171–200 (2006)
77. Yu, K.-M., Zhou, J., Chou, C.-H., Luo, Z.-J., Chen, C.-K.: A Fuzzy Neural Network Based Scheduling Algorithm for Job Assignment on Computational Grids. In: Enokido, T., Barolli, L., Takizawa, M. (eds.) NBiS 2007. LNCS, vol. 4658, pp. 533–542. Springer, Heidelberg (2007)
78. Yu, J., Li, M., Li, Y., Hong, F.: An Economy-Based Accounting System for Grid Computing Environments. In: Web Information Systems – WISE 2004 Workshops, pp. 233–238. Springer, Heidelberg (2004)
79. Zhang, S., Zong, Y., Ding, Z., Liu, J.: Workflow-Oriented Grid Service Composition and Scheduling. In: Proceedings of the International Conference on information Technology: Coding and Computing (Itcc 2005), vol. II, pp. 214–219. IEEE Computer Society, Los Alamitos (2005)

80. Zhou, J., Yu, K.M., Chou, Ch.H., Yang, L.A., Luo, Zh.J.: A Dynamic Resource Broker and Fuzzy Logic Based Scheduling Algorithm in Grid Environment. ICAN-NGA 2007(1), 604–613 (2007)
81. Zomaya, A.Y., Teh, Y.H.: Observations on using genetic algorithms for dynamic load-balancing. IEEE Transactions on Parallel and Distributed Systems 12(9), 899–911 (2001)