Future Generation Computer Systems 26 (2010) 608–621

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Computational models and heuristic methods for Grid scheduling problems

Fatos Xhafa [a,*,1], Ajith Abraham [b]

[a] Department of Computer Science and Information Systems, Birkbeck, University of London, UK

[b] Machine Intelligence Research Labs (MIRLabs), Scientific Network for Innovation and Research Excellence, USA

ARTICLE INFO

Article history:
Received 24 August 2009
Received in revised form
20 November 2009
Accepted 29 November 2009
Available online 4 December 2009

Keywords:
Grid computing
Scheduling
Independent scheduling
Multi-objective optimization
Heuristics
Meta-heuristics

ABSTRACT

In this paper we survey computational models for Grid 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 versus dynamic environment, multi-objectivity, adaptivity, etc., are identified. Then, heuristic and meta-heuristic methods for scheduling in Grids are presented. The paper 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 heuristic and meta-heuristic approaches for the design of efficient Grid schedulers. We also discuss on requirements for a modular Grid scheduling and its integration with Grid architecture.

1. Introduction

Grid computing and Grid technologies primarily emerged for satisfying the increasing demand of the scientific computing community for more computing power. Geographically distributed computers, linked through the Internet in a Grid-like manner, are used to create virtual supercomputers of vast amount of computing capacity able to solve complex problems from e-Science 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 [1–3], Collaborative/e-Science Computing [4,5] and Data-Intensive Computing [6], to name a few.

Grid computing is still in the development stage, and many challenges are to be addressed. Among these, improving its efficiency is a key issue. The question is: *How do we make use of a large number of computers worldwide, ranging from simple laptops, to clusters of computers and supercomputers connected through heterogenous networks in an efficient, secure and reliable manner?*

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 available compatible resources. 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.

An important issue here is how to formally define the Grid scheduling problem. In this paper we present the most important and useful computational models for this purpose. Then, we focus on the design of efficient Grid schedulers using heuristic and meta-heuristic methods. Heuristic and meta-heuristic 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 heuristic and meta-heuristic methods are good alternatives to more traditional scheduling techniques and what make them appropriate for Grid scheduling.

The rest of the paper is organized as follows. We present in Section 2 a few important concepts from Grid computing, and introduce a few types of Grids in view of the needs for different types of scheduling and resource allocation. Then, in Section 3 we identify different types of scheduling problems arising in Grid systems. Computational models for Grid scheduling are given in Section 4, while in Section 5 we focus on the current state of

* Corresponding address: Department of Computer Science and Information Systems, Birkbeck, University of London, Malet Street, London WC1E 7HX, UK. Tel.: +44 0 20 7763 2105; fax: +44 0 20 7631 6727.

E-mail addresses: fatos@dcs.bbk.ac.uk (F. Xhafa), ajith.abraham@ieee.org (A. Abraham).

[1] On leave from Technical University of Catalonia, Spain.

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.11.005

using heuristic and meta-heuristic methods for solving scheduling problems in Grid systems. The integration of Grid schedulers into Grid architecture is tackled in Section 6. A few other issues such as security and Grid services scheduling are discussed in Section 7. We end the paper in Section 8 with some conclusions.

2. The *many* Grids

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 supercomputing for numerical and optimization systems. By the late 1990s, the terms Computational Grids and Grid Computing were popularized by Foster et al. [7], who developed the Globus Toolkit as a general middleware for Grid Systems. Since then, Grid systems have advanced very quickly. In the following subsections we briefly review the most important types of Grids that have appeared during recent years.

Computational Grids. Computational Grids are among the first type of Grid systems. They were developed due to the need to solve problems that require processing a large quantity of operations or data. In spite of the fact that the capacity of the computers continues to improve, the computational resources do not respond to the continuous demand for more computational power. Moreover, many statistical studies have shown that computers from companies, administration, etc. are usually underutilized. One of the main objectives of the Computational Grid is, therefore, to benefit from the existence of many computational resources through the sharing.

Scavenging Grids. In such Grids, the politics of "*scavenging*" is applied, according to which, 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. Sethi@home project is an example of scavenging Grids.

e-Science Grids. Under the name of e-Science 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 infrastructure (access to computational and data resources) needed to solve many complex problems arising in areas of science and engineering. Representative examples are EGEE Grid Computing, UK e-Science Grid, German D-Grid, BIG GRID (the Dutch e-Science Grid) and French Grid'5000, among others.

Data Grids. Data Grids 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. In such Grid systems many types of algorithm, such as replication, are important to increase the performance of Grid-enabled applications that use large amounts of data. Also, data copy and transfer is important here in order to achieve high throughput.

Enterprise Grids. Nowadays Grid computing is becoming a significant component of business as well. Indeed, today's e-business must be able to respond to increasing consumer demands and adjust dynamically and efficiently to marketplace shifts. Enterprise Grids enable running several projects within one large enterprise to share resources in a transparent way. Enterprise Grids are thus showing great and innovative changes on how computing is used. 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".

Desktop Grids. A new form of Enterprise Grids is also emerging in institutions, the so-called Desktop Grids, which use the idle cycles of desktop PCs. 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 the low heterogeneity and volatility of resources. 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.

3. Scheduling problems in Grid systems

Rather than a problem, scheduling in Grid systems is a family of problems. This is due to the many parameters that intervene scheduling as well as due to the different needs of Grid-enabled applications. In the following, we give some basic concepts of scheduling in Grid systems and identify the 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 [8].

3.1. Basic concepts and terminology

Although many types of resources can be shared and used in a Grid system, usually 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 subtasks; the Grid system is responsible for sending each subtask to a resource to be solved. In a simpler Grid scenario, it is the user who selects the most adequate machine to execute its application or subtasks. However, in general, Grid systems must dispose of *schedulers* that automatically and efficiently find the most appropriate machines to execute an assembly of tasks.

3.1.1. New characteristics of scheduling in Grids

The scheduling problem 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 and more challenging than its version of conventional distributed systems. Some of these characteristics are the following.

- *The dynamic structure* of the Computational Grid. Unlike traditional distributed systems, resources in a Grid system can join or leave the Grid in an unpredictable way. This could be simply due to losing 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*. In Grid systems, computational resources could be very disparate in their computing capacity, ranging from laptops, desktops, clusters, supercomputers and even small computational devices. Current Grid infrastructures are not yet much versatile but heterogeneity is among most important features in any Grid system.
- The *high heterogeneity of jobs*. Jobs arriving at any Grid system are diverse and heterogenous in terms of their computational

needs. For instance, they could be computing intensive or data intensive; some jobs could be full applications having many specifications and others could be just atomic tasks. Importantly, in general the Grid system will not be aware of the type of tasks arriving in the system.

- The *high heterogeneity of interconnection networks.* Grid resources are connected through the 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.* 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 system. In such cases, one possible requirement would be to use the local scheduler.
- 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 on access, available storage, *pay-per-use*, etc. are also to be taken into account.
- The *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.
- The *large scale of the Grid system*. Grid systems are expected to be large scale. Similarly, the number of jobs, tasks or applications submitted to the Grid over time could be large as well. 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 to achieve scalability.
- *Security*. This characteristic, which is non-existent 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 security requirements and, on the other hand, the Grid nodes could have their own security requirements.

3.1.2. 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 availability, performance criteria and produce a planning of jobs 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) or there could be dependencies (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 the job description. In the simplest case, a job could have just one task.

Application: An application is the software for solving a problem in a computational infrastructure; it may require splitting the computation into 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. Applications could have different resource requirements and constraints, usually expressed within the 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. Moreover, the resources may belong to different administrative domains, implying different policies on usage and access.

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 [9].

Resource pre-reservation: Pre-reservation is needed either when tasks have requirements on the finishing time or when there are dependencies 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 to computational resources.

Grid scheduler: Software components in charge of computing a mapping of tasks 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-schedulers, local/cluster schedulers and enterprise schedulers. As a main component of any Grid system, the 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 coexist, and they could in general pursue conflicting goals; thus, there is the need for interaction and coordination between the different schedulers in order to execute the tasks.

Super-scheduler: This kind of scheduler corresponds to a centralized scheduling approach in which local schedulers are used to reserve and allocate resources in the Grid, while the local schedulers manage their job queue processing. The super-scheduler is in charge of managing the advance reservation, negotiation and service level agreement.

Meta-scheduler: This kind of scheduler (also known as a meta-broker) arises 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 here.

Local/cluster scheduler: This kind of scheduler is in charge of assigning tasks to resources in the same local area network. The scheduler manages the local resources and the local job queuing system and is thus 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 immediate mode scheduling, tasks are scheduled as soon as they enter the system.

Batch mode scheduling: In batch mode scheduling, tasks 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, 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.

Cooperative scheduling: In cooperative scheduling, a feasible schedule is computed through the cooperation of procedures, rules, and Grid users.

High-throughput schedulers: The objective of this kind of scheduler [10] 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 scheduler is concerned with scheduling applications in order to meet a 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.

3.1.3. 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 submitted to the Grid; (2) Resource selection; (3) Computation of the planning of tasks to selected resources; (4) Task (job or application) allocation according to the planning (the mapping of tasks to selected resources); and (5) Monitoring of task completion (the user is referred to [11] for a detailed description).

Preparation and information gathering: Grid schedulers have access to the information on available resources and tasks through the Grid Information Service. Moreover, the scheduler will be informed about updated information (according to the scheduling mode) on jobs and resources.

Resource selection: Not all resources could be candidates for the allocation of tasks. 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 user specifications.

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

Task allocation: In this phase the planning is made effective: tasks 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 or migrated to other resources.

3.2. Types of scheduling in Grids

Different types of scheduling are found in Grid systems as applications could have different scheduling needs such as batch or immediate mode, task independent or dependent; on the other hand, the Grid environment characteristics themselves impose restrictions such as dynamics, use of local schedulers, centralized or decentralized approach, etc. It is clear that in order to achieve the desired performance, both the 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 computational Grids. Aside from the purely scientific needs, the computational power is causing changes in important industries such as biomedics, oil exploration, digital animation, aviation, financial fields, and many others. 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), which can be scheduled independently.

Grid workflows. Solving many complex problems in Grids requires 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 is known as Grid workflows. Such applications can take advantage of the power of Grid computing; however, the characteristics of the Grid environment make the coordination of its execution very complex [12,13]. Besides the efficiency, Grid workflows should deal with robustness. Certainly, a Grid workflow could run for a long period, which in a dynamic setting increases the possibility of process failure, causing failure of the whole workflow, if failure recovery mechanisms are not used.

Centralized, hierarchical and decentralized scheduling. Both centralized and decentralized scheduling are useful in Grid computing. Essentially, they differ in the control of the resources and 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 resource state, and therefore it is easier to obtain efficient schedulers. This type of scheduling, however, suffers from limited scalability and is thus not appropriate for large-scale Grids. Moreover, centralized schedulers have a single point of failure. Another way to organize Grid schedulers

is hierarchically, which allows one to coordinate different schedulers at a certain level. In this case, schedulers at the lowest level in the hierarchy have knowledge of the resources. This scheduler type still suffers from lack of scalability and fault tolerance, yet it scales better and is more fault tolerant than centralized schedulers. In decentralized or distributed scheduling there is no central entity controlling the resources. The autonomous Grid sites make it more challenging to obtain efficient schedulers. In decentralized schedulers, the local 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 job queue. 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 versus dynamic scheduling. There are essentially two main aspects that determine the dynamics of the Grid scheduling, namely: (a) *The dynamics of job execution*, which 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; and (b) *The dynamics of resources*, in which 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. These two factors decide the behavior of the Grid scheduler, ranging from static to highly dynamic. For instance, in the static case, there is no job failure and resources are assumed available all the time (e.g. in Enterprise Grids). Although this is unrealistic for most 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 versus batch mode scheduling. Immediate and batch scheduling are well-known methods, largely explored in distributed computing. 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 are grouped in batches and scheduled as a group. In contrast to immediate scheduling, batch scheduling could take better advantage of job and resource characteristics in deciding which job to map to which resource since they dispose of the time interval between two successive activations of the scheduler.

Adaptive scheduling. The changeability over time of the Grid computing environment requires adaptive scheduling techniques [14], which will take into account both the 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 migrated to more suitable resources. Casanova et al. [15] 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 [1] as a testbed for evaluating the proposed algorithm. Othman et al. [16] 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. [17] reported a scheduling algorithm built on top of the GridWay framework, which uses internally adaptive scheduling.

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 the 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 a type of scheduling. In fact, the usefulness of the 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, in general, data should be "close" to tasks to achieve efficient access.

4. Computational models for Grid scheduling

Given the versatility of scheduling in Grid environments, one needs to consider different computation models for Grid scheduling that would allow one to formalize, implement and evaluate – either in a real Grid or through simulation – different scheduling algorithms. We now 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 are submitted for completion to a single resource.

4.1. Expected time to compute model

In this model [18], 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 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 [19].

Modelling heterogeneity and consistency of computing. The *ETC* matrix model is able to describe different degrees of heterogeneity in a 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 the 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. In contrast, 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 submatrix 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 shortly, it is almost straightforward to define several optimization criteria in this model to measure the quality of a schedule. It is worth noting that incompatibilities among tasks and resources can also be expressed in the 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.

4.2. Total processor cycle consumption model

Despite its interesting properties, the ETC model has an important limitation, namely, the computing capacity of resources is assumed unchanged during task computation. This limitation becomes more evident when we consider Grid systems in which not only do the resources have different computing capacities but also they could change over time due to the 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 [20] 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 the ETC model, the task workload is expressed in number of instructions and the computing capacity of resources in number of instructions computed per unit time. The total consumption of computing power due to Grid application completion is measured. Clearly, this model takes into account that resources could change their computing speed over time, as happens in large-scale computing systems whose workload is in general unpredictable.

Problem instance. A problem instance in the TPCC model consists of the vector of task workloads (denoted task lengths in [20]) 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. Then a matrix *PS* (for Processor Speed) is built over time 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.

4.3. 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 (GIS) 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 GIS. 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 First-Come First-Served, Earliest Deadline First, 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 the 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 across different clusters. Clusters could belong to different enterprises and institutions; that is, to 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 model 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 the throughput for user tasks. This model has been exploited in Lee and Zomaya [21] 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.

4.4. Grid system performance and optimization criteria

Several performance requirements and optimization criteria can be considered for Grid scheduling—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 one 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, waiting time and response time. In fact other criteria could also be considered for characterizing a 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 could be conflicting; for instance, minimizing makespan conflicts with resource usage and response time.

One of the most popular and extensively studied optimization criteria 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 does not necessarily imply the optimization of other objectives. As mentioned above, its optimization could in fact be to the detriment of 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 implies 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. Makespan indicates 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*: $\min_{S_i \in Sched}\{\max_{j \in Tasks} F_j\}$, and
- minimization of *flowtime*: $\min_{S_i \in Sched}\{\sum_{j \in Tasks} F_j\}$,

where F_j denotes the time when 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 the flowtime, tasks should be executed in ascending order of their workload.

The completion time of a machine m is defined as 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, by first sorting the tasks in ascending order of their *ETC* values. Thus for machine m, $flowtime[m]$ 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 the TPCC model, for a schedule S of makespan M, the Total Processor Cycle Consumption 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 the TPCC value and makespan value; however, the longer the makespan, the larger the value of TPCC, and vice versa. In other words, it could be established that any schedule with a good TPCC value is a schedule with a good makespan value also. 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 the TPCC model is appropriate not only for heuristic-based scheduling methods without guarantee of fitness value of the TPCC but also for *approximation*[2]-based schedulers ensuring a quality of delivered schedule.

[2] 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.

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 increase the benefit of the resource owners, the scheduler should use all resources, yet this could contradict 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 QoS 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:

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

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

Matching proximity. This metric 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. Expressing this criterion explicitly is more difficult, as compared to other criteria.

In the ETC model, the matching proximity could be defined as the degree of proximity of a given schedule with regard to the schedule produced by the *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, the 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. [22] considered this objective for scheduling bag-of-tasks applications in desktop Grids. This objective is usually more important in batch scheduling than in interactive applications. Kondo et al. [23] 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 a user's tasks have priorities. Usually, this criterion is implemented through weights associated to the tasks [24,25]:

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

where w_j is the priority (weight) of job j and F_j denotes the time when the task j finalizes. As in the case of flowtime, this criterion can be seen as a QoS to the Grid user. In a similar way the *total weighted tardiness* and the *weighted number of tardy jobs* are defined for the case of tasks 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 task j, F_j its finalization time and R_j its submission time to the Grid system. This criterion can then be expressed as follows:

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

where $(F_j - R_j)$ is the response time of task j. In [26,27], the response time of a task is weighted by its resource consumption (long jobs have larger resource consumption than short jobs) to balance the impact of short jobs versus long jobs with a higher resource consumption.

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

4.5. 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 such 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 approaches.

Hierarchical approach. This approach is useful to establish priority among the criteria. For instance, in high-performance computing we could give more priority to the makespan and less priority to the response time; yet, if the user requirements are involved, we could consider the reverse priority. Let c_i, $1 \leq i \leq k$ 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 criterion c_j cannot be varied while optimizing according to c_i. This approach is especially useful when the criteria are measured in different units and cannot be combined in a single aggregate objective function (for instance, optimizing makespan and the number of tardy tasks). This approach has been considered in Xhafa [28,29] for independent job scheduling under the 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 certainly has a high computation cost. It should be addressed through Pareto optimization theory [30,31]. However, in the Grid scheduling problem, rather than knowing many Pareto points in solution space, we could be interested in knowing a schedule having a tradeoff among the considered criteria and which could be computed very quickly. Therefore, we can consider a small number of objectives at the same time, which in general suffices for practical applications.

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, meta-heuristic and hybrid approaches for single-objective problems. There are two issues here: first, how to combine the different objectives in a meaningful way in a single-objective function – in fact this is not always possible – and, second, how to find suitable values for the weights of the criteria, which *per se* introduces new variables to the problem definition. For practical cases, however, one could fix *a priori* the weights either based on 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 [30,31]. Many classes of meta-heuristic algorithms have been developed for multi-objective optimization, e.g., the Multi-objective Genetic Algorithms (MOGA) [32].

As an example let us consider the case (a) when makespan and flowtime are considered simultaneously. As mentioned 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. [33,28,29,34] the value of λ is fixed, based on preliminary tuning, to $\lambda = 0.75$; that is, more priority is given to makespan. In many meta-heuristic implementations, it was observed that this single aggregate objective function shows good performance and outperforms known approaches in the literature for independent Grid scheduling.

5. Heuristic and meta-heuristic methods for scheduling in Grids

From the exposition in the previous sections, it is clear that the 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 undoubtedly considered the *de facto* approach. We now point out the main reasons that explain the strength of meta-heuristic approaches for designing efficient Grid schedulers.

- *Meta-heuristics are well understood*: 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-heuristic-based Grid schedulers.
- *"No need" for optimal solutions*: In the Grid scheduling problem, for most practical applications good quality planning of jobs would suffice rather than searching for optimality. In fact, in the highly dynamic Grid environment, it is not possible to even define the 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 exists 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 a short time high-quality feasible solutions. Therefore, in such situation meta-heuristics are among best candidates to cope with Grid scheduling.
- *Efficient solutions in short time:* Research work on meta-heuristics has by large tried to find ways to avoid getting stuck in local optima and ensure convergence to suboptimal or optimal solutions. However, meta-heuristics dispose of mechanisms that allow one to tune 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 the Tabu Search method, one can work with just short-term memory (recency) in combination with an intensification procedure to produce high-quality feasible solutions in a 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 have proven to efficiently solve not only single-objective optimization problems but also multi-objective optimization problems.
- *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 the online bank's clients' transaction activity. In this case 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-heuristic-based schedulers for longer execution times and significantly increase the quality of planning of jobs to resources. Similarly, in batch scheduling, we could run the meta-heuristic-based scheduler for the time interval comprised within two successive batches activations.
- *Appropriateness for decentralized approaches:* Since Grid systems are expected to be large scale, decentralization and coexistence of many Grid schedulers in the system is desirable. We could thus have many instances of the meta-heuristic-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 Grid types, specific types of applications, etc. The hybridization in general can produce better solutions than those delivered by single 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 should deliver high-quality planning even under frequent changes of the characteristics of the Grid infrastructure. Evidence in meta-heuristics literature exists that in general meta-heuristics are robust approaches.
- *Libraries and frameworks for meta-heuristics:* Many libraries and frameworks have been developed in the literature for meta-heuristics, for both single-objective and multi-objective cases. For instance, the Mallba library [35], Paradiseo [36] and EasyLocal + + [37] are such libraries. These libraries can be easily used for the Grid scheduling problem; for instance, the meta-heuristic approaches in Xhafa et al. [33,28] used skeletons defined in the Mallba library.

In the following subsections we briefly review the most important heuristic and meta-heuristic approaches and the benefits of using them for the Grid scheduling problem (the reader is referred to [38,39] for a survey on meta-heuristic approaches).

5.1. Local search-based heuristic approaches

Local search [40] is a family of methods that explore the solution space by starting at an initial solution, and construct a path in solution space during the search process. Methods in this family include Hill Climbing (HC), Simulated Annealing (SA) and Tabu Search (TS), among others.

Simple local search methods (HC-like) are of interest, at least, for two reasons: (1) they produce a feasible solution of certain quality within a very short time; and (2) they can be used to feed (initialize) population-based meta-heuristics with genetically diverse individuals. Such methods have been studied for the scheduling under the ETC model in Ritchie and Levine [41]. Xhafa [29] used several local search methods in implementing Memetic Algorithms (MAs) for the same problem.

SA is more powerful than simple local search by accepting also worse solutions with certain probability. This method has been proposed for Grid scheduling by Abraham et al. [42] and Yarkhan and Dongarra [43].

TS [44] is more sophisticated and usually requires more computation time to reach good solutions. However, its mechanisms of tabu lists, aspiration criteria, intensification and diversification make it a very powerful search algorithm. Abraham et al. [42] considered TS for the problem. Ritchie [45] implemented a TS for the problem under the ETC model and used it in combination with an ACO approach. An ant approach is reported also in [46]. Recently, Xhafa et al. [47] have presented the design, implementation and evaluation of a full TS for the scheduling under the ETC model, which outperformed Ritchie's approach.

We now present the design of simple local search methods for the Grid scheduling problem.

Design of local search methods for Grid scheduling. Simple local search methods can be applied straightaway to the Grid scheduling. In fact, many variations of these methods can be designed by considering different neighborhood structures as well as ways 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 the minimization 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 differ only 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 its resource to another randomly chosen resource; (b) *Steepest Local Move (SLM)* moves a randomly chosen task to the resource yielding the largest improvement in optimization criteria; (c) *Local MCT Move (LMCTM)* is based on the MCT (*Minimum Completion Time*) heuristic. In LMCTM, a task is moved to the resource yielding the smallest completion time among all the resources; (d) *Local Minimum Flowtime Move (LMFTM)* moves a task that yields the largest reduction in the flowtime.
- *Swap-based local search*: In this group of methods, the neighborhood is defined 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 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 process of *Short Hop* [48]. 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 for the neighborhood definition. We can thus design: (a) *Local Rebalance (LR)*: the movement from a solution to a neighboring one is done by reducing the load of the most loaded resources; (b) *Deep Local Rebalance (DLR)*: applies a movement with the largest improvement in rebalancing; (c) *Local Flowtime Rebalance (LFR)*: swaps 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; (d) *Emptiest Resource Rebalance (ERR)*: in this method the aim is to balance the workload of the resources but now the less loaded resource

is used; and (e) *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. 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).

5.2. Population-based heuristic approaches

Population-based heuristics is a large family of methods that have shown their efficiency for solving combinatorial optimization problems. Population-based methods usually require large running times if suboptimal 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 the case of Grid scheduling, we can exploit the inherent mechanisms of these methods to increase the *convergence* of the method.

We could distinguish three 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. [42], Braun et al. [48], Zomaya and Teh [49], Martino and Mililotti [50], Page and Naughton [51], Gao et al. [52] and Xhafa et al. [28,53].

MAs [54] is a relatively new class of population-based methods, which combine the concepts of evolutionary search and local search. 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 the Grid scheduling problem. Xhafa [29] applied unstructured MAs and Xhafa et al. [34] proposed Cellular MAs (structured MAs) for independent scheduling under the ETC model.

An ACO implementation for the problem under the ETC model has been reported by Ritchie [45]. Abraham et al. [55] proposed an approach for scheduling using a fuzzy PSO algorithm.

Specific methods for population initialization. In population-based methods, it is important to dispose a wide variety of initialization methods for the generation of 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. We distinguish 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) [56,57,48,42].

5.3. Hybrid heuristic approaches

Meta-heuristic methods are *per se* hybrid approaches. For instance, MAs combine an evolutionary search with a local search. However, hybridization among different meta-heuristics has been shown to be effective for many problems by outperforming single methods [58]. However, hybrid meta-heuristics have been less explored for the problem. Abraham et al. [42] addressed the hybridization of GA, SA and TS heuristics; GA+SA hybridization is expected to have a better convergence than a pure GA search and GA+TS could improve the efficiency of the GA. In these hybridizations a heuristic capable to deal with a population of solutions, such as a GA, is combined with 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 [59], who combine an ACO algorithm with a TS algorithm for the problem. In [29], a basic unstructured MA is combined with several local search algorithms in order to identify the best performance of the resulting MA.

5.4. Other approaches

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

Hyper-heuristic approaches. Hyper-heuristics [60] are methods that guide the search, at a higher level as compared to meta-heuristics. Hyper-heuristics have proven to be effective for scheduling and timetabling (Burke et al. [61]) and are therefore also candidate approaches for Grid scheduling.

Xhafa [62] presented a simple hyper-heuristic for the problem, which uses as underlying heuristics a set of *ad hoc* (immediate and batch mode) methods for 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, 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.

Reinforced learning. Perez et al. [63] proposed implementing Reinforcement Learning for scheduling in large Grid systems. Vengerov [64] presented a utility-based framework for making repeated scheduling decisions dynamically; the observed information about unscheduled jobs and the system's resources is used for this purpose.

Fuzzy logic, neural networks and QoS approaches. Zhou et al. [65] used fuzzy logic (FL) techniques to design an adaptive FL scheduler, which utilizes the FL control technology to select the most suitable computing node in the Grid environment. A Fuzzy Neural Network (NN) was proposed by Yu et al. [66] to develop a high-performance scheduling algorithm. The algorithms uses FL techniques to evaluate the Grid system load information, and adopts the NNs to automatically tune the membership functions. Hao et al. [67] presented a Grid resource selection based on NNs aiming to achieve QoS. To this end, the authors propose selecting Grid resources constrained by QoS criteria. The resource selection problem is solved using NNs.

Economy-based scheduling. Economy-based models are important for the design of resource management architecture for Grid systems. Several recent works [68–71] address 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.

Game-theoretic based scheduling. Game-theoretic models are showing their usefulness in the analysis and design of distributed computing and networking algorithms. In particular, there has been an increasing interest in using game-theoretic models for Grid scheduling [72]. Recently, Kolodziej and Xhafa [73] have proposed a game-theoretic and GA model for security-assured scheduling of independent jobs in Computational Grids.

Grid services scheduling. W3C defined a service as a set of actions that form a coherent whole from the point of view of

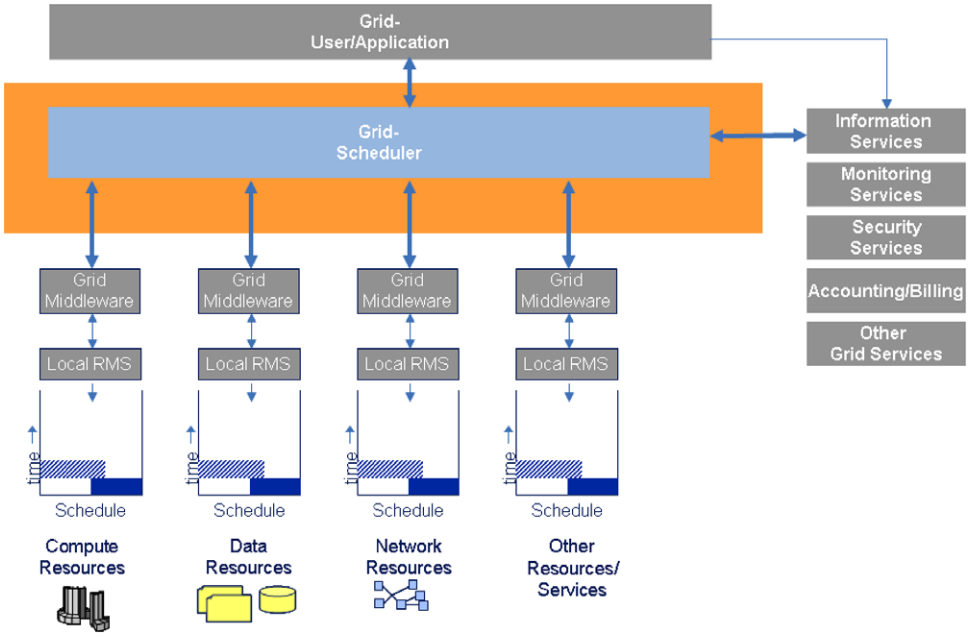

Fig. 1. Scheduling architecture [77].

service providers and service requesters. Although this definition originated for web systems, services were defined similarly for Grid systems. There are two aspects 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 works [74–76] explore these aspects, yet there is still little research work in this direction.

6. Integration of schedulers into Grid architecture

Job scheduling technologies have made it possible to achieve the vision of the high-performance Grid by making tractable many computationally hard problems. Scheduling components, although crucial and useful components of a Grid, are just part of a larger system. The complete vision of a Grid is delivered from the combination of several Grid technology domains that achieve the *virtualization*: workload virtualization, information virtualization, system virtualization, storage virtualization, provisioning, negotiation, and orchestration. Therefore, schedulers are just components of a modular scheduling architecture, coupled with the Grid middleware that provides access to independent local computational resource managers [77] (see Fig. 1).

Basic components of Grid scheduling architecture.

The following components have been identified as building blocks of Grid scheduling architecture [77]:

- *Scheduling service*: queries for resources and computes a planning of job(s) to available resources.
- *Job supervisor service*: monitors job executions.
- *Information service*: includes both static and dynamic information on resources, data, network state, etc.
- *Data management service*: provides information services with information on data.
- *Network management service*: provides information services with information on networks.
- *Accounting and billing service*: provides budget information for allocations, etc.

The integration of schedulers into the Grid architecture represents one of the major efforts and has thus been addressed in many Grid computing projects. We exemplify next the integration approach through two most paradigmatic Grid computing projects, namely Globus and EGEE Grid Computing. Other examples include UNICORE/ARC [78] (from NorduGrid).

The Grid Resource Allocation Services of Globus Grid. Grid Resource Allocation Services Package (GRASP) is part of the Globus Toolkit in charge of resource management. GRASP is actually seen as an upper level of job submission and scheduling system (see Fig. 2).

gLite Workload Management System of EGEE Grid Computing. Enabling Grids for E-sciencE (EGEE) is a large, EU-funded Grid computing project. The EGEE project provides large-scale computing infrastructure for researchers conducting studies in data-intensive and computing-intensive applications from high-energy physics, earth and life sciences.

One key part of the EGEE Grid project is the gLite middleware, which includes also the Workload Management System (WMS). The gLite WMS [79] (see Fig. 3 for its architecture) can be seen as a collection of components responsible for distributing and managing users' jobs onto computing and storage of available Grid resources.

The interfaces of gLite WMS facilitate the submission process of the end-user, on the one hand, by hiding the complexities of communicating with a highly heterogeneous and dynamic infrastructure, and on the other, by managing task recovery in the case of failure. Several job types, such as batch, DAG workflow, parametric and interactive, are supported by gLite WMS. The gLite WMS has been tested very intensively for ATLAS project experiments (simulated data for physics studies), and showed good performance and reliability [80].

7. Further issues

Besides the many aspects and facets of the Grid scheduling problem presented in the previous sections, there still remain other issues to be considered. We briefly mention some of them here.

Security is an important aspect to be considered in Grid scheduling. It can be seen as a two-fold objective: tasks could have security requirements to be allocated in secure nodes, while the node itself could have security requirements; that is, the tasks

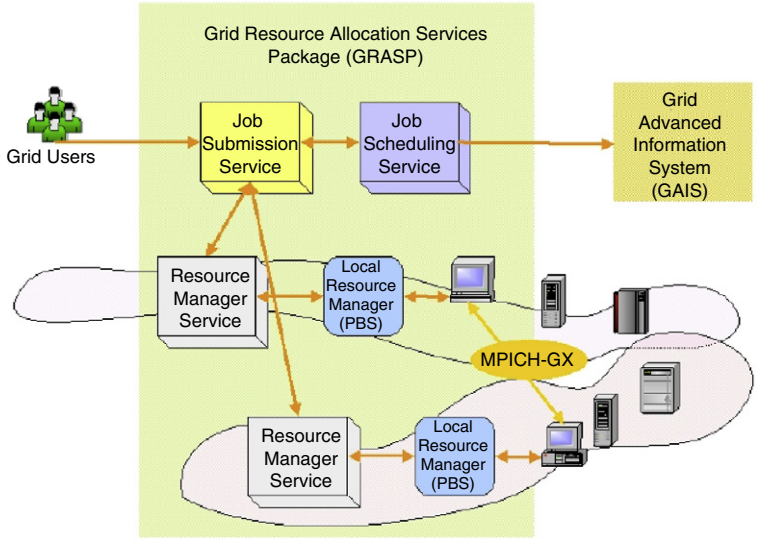

Fig. 2. GRASP architecture (from the Globus Toolkit).

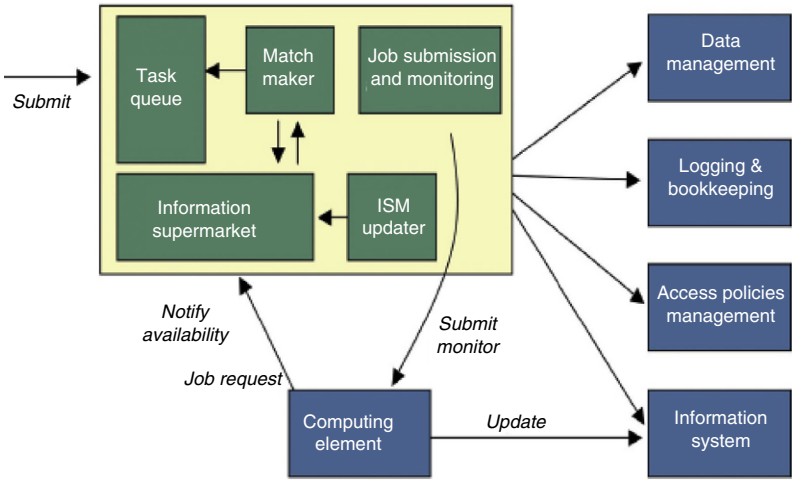

Fig. 3. The gLite workload management system (from the EGEE Grid Computing project).

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 range from very untrustworthy to very trustworthy scale. Moreover, the objective is to reduce the possible overhead to the Grid scheduler that would introduce a secure scheduling approach.

Other important issues are related to data-aware scheduling [81]. Most current Grid approaches are task-oriented or resource-oriented approaches. For instance, tasks are assumed to include all data needed for its computation or tasks are just the processes and data is assumed to be available in Grid nodes. However, with the ever-increasing complexity of large-scale problems in which both tasks and data are to be scheduled, an integrated scheduling approach that would optimize allocation of both the task and the data is required.

8. Conclusions

In this paper, we have surveyed the most important concepts from Grid computing related to scheduling problems, their resolution using heuristic and meta-heuristic approaches and the integration of Grid schedulers into Grid architectures. 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 versus dynamic environment, multi-objectivity, adaptivity, etc. Our study revealed the complexity of the scheduling problem in computational Grids when compared to scheduling in classical parallel and distributed systems and shows the usefulness of heuristic and meta-heuristic 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.

Acknowledgements

Fatos Xhafa's research work was partially done at Birkbeck, University of London, while on leave from the Technical University of Catalonia (Barcelona, Spain). His research is supported by a grant from the General Secretariat of Universities of the Ministry of Education, Spain.

References

[1] H. Casanova, J. Dongarra, Netsolve: Network enabled solvers, IEEE Computational Science and Engineering 5 (3) (1998) 57–67.