Resource Selection for Tasks with Time Requirements Using Spectral Clustering

Nikolaos D. Doulamis, *Member*, *IEEE*, Panagiotis Kokkinos, and Emmanouel (Manos) Varvarigos

Abstract—Resource selection and task assignment are basic operations in distributed computing environments, like the grid and the cloud, where tasks compete for resources. The decisions made by the corresponding algorithms should be judged based not only on metrics related to user satisfaction, such as the percentage of tasks served without violating their quality-of-service (QoS) requirements, but also based on resource-related performance metrics, such as the number of resources used to serve the tasks and their utilization efficiency. In our work, we focus on the case of tasks with fixed but not strict time requirements, given in the form of a requested start and finish time. We propose an algorithm for assigning tasks to resources that minimizes the violations of the tasks' time requirements while simultaneously maximizing the resources' utilization efficiency for a given number of resources. The exact time scheduling of the tasks on the resources is then decided by taking into account the time constraints. The proposed scheme exploits concepts derived from graph partitioning, and groups together tasks so as to 1) minimize the time overlapping of the tasks assigned to a given resource and 2) maximize the time overlapping among tasks assigned to different resources. The partitioning is performed using a spectral clustering methodology through normalized cuts. Experimental results show that the proposed algorithm outperforms other scheduling algorithms for different values of the granularity and the load of the task requests.

Index Terms—Resource assignment, spectral clustering, graph partitioning, interval scheduling, soft time constraints

1 Introduction

There are several ways to express the quality-of-service (QoS) requirements of the tasks submitted to a distributed computing environment, such as the grid and the cloud [1]. The most common way to describe QoS in a grid or cloud environment is through task deadlines. However, if QoS is defined through deadlines alone, there is no direct relation to the time period a task uses a resource and/or to the price the corresponding user will be charged for this use, an important consideration in commercial distributed computing infrastructures. For example, a task with a late deadline may require short execution time, while a task with an imminent deadline may correspond to a heavy computation workload, and vice versa.

An alternative way to express the QoS requirements is through a task's start time, duration, and dependencies on other tasks. However, in real-life applications, predicting the execution time of a task is not a straightforward process, even though some related works exist for specific application domains (e.g., in [2] for 3D image rendering).

A different approach is to specify the tasks' QoS by their requested start and finish times [3]. In such a description, users know a priori the time period during which they need a resource, or the amount they can afford to pay for it (which translates directly to time) and the question is to select the resources they should utilize. It is often both acceptable and expected that the actual time interval allocated to a task may differ, to some degree, from the requested one. This is because of two problems: 1) the tasks compete for resources with each other and 2) a task often takes more execution time than the requested duration. In this paper, we propose an optimal resource selection strategy for addressing the first, the task contention, problem in a grid or cloud environment where each task requests a given time interval (defined by its start and end times) for execution. In case a task does not complete execution within this time period, preemption is a common approach, where the remaining task workload is assigned to another resource interval.

Such a QoS definition fits particularly well with the cloud computing environment. In cloud computing, virtual services are created on the fly and schemes that perform in advance timed reservations are often used to guarantee service execution on virtual machines [4], [5]. When a task's QoS is defined as its requested start and finish time, it is directly related with the time reservations of the resources and, thus, with the price the user is expected to pay. Alternative QoS measures can usually be obtained easily, once the start and finish times are given, and can, therefore, be considered of a secondary nature.

1.1 Previous Approaches

The primary objective of most existing scheduling approaches is to improve system resource utilization, while the quality of service experienced by grid/cloud users is, at best, a secondary consideration [6], [7]. Still, several QoS scheduling algorithms have been reported in the literature, using different QoS definitions and, thus, trying to achieve different objectives. In Section 1.1.1, we briefly describe the state of the art in case the task QoS requirements are defined through task deadlines and durations. In Section 1.1.2, we focus on corresponding algorithms for the case where the tasks' QoS requirements are defined through their requested start and finish times.

- *N.D. Doulamis is with the National Technical University of Athens, 9, Heroon Polytechniou Str., 157 73 Zografou, Athens, Greece. E-mail: ndoulam@cs.ntua.gr.*
- *P. Kokkinos and E. Varvarigos are with the Department of Computer Engineering and Informatics, University of Patras, Patras University Campus, Building B, Rio, Patras 26504, Greece. E-mail: {kokkinop, manos}@ceid.upatras.gr.*

Manuscript received 8 Nov. 2010; revised 24 Aug. 2012; accepted 28 Aug. 2012; published online 11 Sept. 2012.
Recommended for acceptance by J. Weissman.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-11-0610.
Digital Object Identifier no. 10.1109/TC.2012.222.

1.1.1 General Scheduling Algorithms

When the tasks' QoS requirements are given in the form of task deadlines, the most well-known scheduling algorithm is the Earliest Deadline First (EDF) [8]. Another approach is the Slack Time algorithm, also referred to as Least Laxity First (LLF) [9], where the tasks are selected for execution in order of nondecreasing slack time, defined as the difference between the task's relative deadline and its remaining computational time. A framework for providing hard deadline guarantees in a grid environment, by controlling the rate and burstiness with which users submit jobs to resources, is proposed in [10].

In the case where the infrastructure consist of a cluster of servers, several schedulers have been developed, including Maui [11], the portable batch system (PBS) [12], and the load sharing facility (LSF) [13]. Most of the scheduling algorithms proposed so far for grids provide a best effort (nonguaranteed) service to the submitted tasks [14] and try to optimize a single or multiple metric(s) of interest (for example, task delay, probability to miss a deadline, etc.). Scheduling algorithms also exist that take into account the users' QoS requirements (delay, required number of CPUs, etc.) and try to serve them in the best possible way [15]. Fairness in grid scheduling has been considered in [16], [17], where the proposed algorithms allow for the "fair" sharing of deadlines' violations among all users. Genetic algorithms have also been widely used for solving NP-complete scheduling problems; see, for example, the works of [18], [19], [20], [21].

Reservation algorithms have been studied in many works [1], [22], [23], as a way to provide QoS guarantees. The Globus Architecture for Reservation and Allocation (GARA) [24] and the grid quality-of-service management (G-QoSm) framework [25] are such examples. The scalability problem of resource selection algorithms is addressed in [26], where it is shown that decoupling resource selection from time scheduling is beneficial.

QoS and workflow issues have also been investigated in the literature. Related works usually model dependencies through Directed Acyclic task Graphs (DAG), and provide scheduling algorithms for heterogeneous or homogeneous computing environments [27], [28], [29], [30], [31]. Performance results for such algorithms are presented in [32].

1.1.2 Interval Scheduling

Our work relates to a particular flavor of the scheduling problem, which is called the interval scheduling problem, also known as fixed job scheduling [33]. In interval scheduling, a list of tasks is given as a set of requested time intervals and the scheduler decides if it can accept or not a task, and, in the second case, assign it to some resource. There are several variants of the problem:

- One variant assumes an unrestricted number of resources and the objective is to find a minimum-cost schedule that does not reject any tasks [34].
- Another variant assumes a fixed number of resources and a predefined profit associated with the successful execution of each task. The objective is to find a maximum-profit schedule in which some, but not necessarily all, of the tasks are assigned to the resources [3].
- The third case refers to a variant of the aforementioned problems where each task has a set of possible starting times, instead of a single starting time [35].
- Finally, in the online interval scheduling problem tasks are generated one by one and the scheduler has to make a decision for each task before a new task arrives. The objective is to maximize the total length of the intervals assigned to tasks, while ensuring that no pair of tasks assigned to the same resource overlap in time [36].

In [33], the interested reader can find a survey of the interval scheduling literature. In the current paper, we focus on the second type of interval scheduling algorithms. This variant of the problem can be modeled using a min-cost flow formulation (see [3], [37]). In the case that all jobs have unit weight, greedy algorithms are proposed in [38], [39] for estimating the maximum number of jobs that can be served. Heuristic and exact algorithms have been proposed in [40] for the case where each job can be executed only on a given subset of machines.

The interval scheduling problem can also be formulated using interval graphs. A node in the interval graph corresponds to a task and there is an edge between nodes whose task intervals overlap. Hence, the basic interval scheduling problem is in fact a coloring problem of the interval graph (see [41]). Interval graph scheduling can also be formulated as a graph partitioning problem, where the interval graph is partitioned so as to satisfy a multiobjective criterion. Graph partitioning is a kind of task clustering. Task clustering has been used in the literature for task scheduling in grids. Examples can be found in [42], [43], where two task routing policies (one static and one adaptive) and six task scheduling policies are examined. Methods with similar goals include workflow allocation, mixed local, and global scheduling [44], [45].

1.2 Contribution

Tasks generally compete for resources, and the situation often arises that they cannot all be assigned to the limited resources without overlapping in time. The Interval scheduling approaches mentioned in Section 1.1.2 handle this situation by rejecting the overlapping tasks or assigning them to a subsequent scheduling period. This increases delays and may cause cascading effects in case of dependent tasks, making resource assignment a challenging problem. Additionally, this is not a fair policy, since tasks submitted by users that pay the same price, or contribute equally to a common infrastructure and should, therefore, have the same priority, are handled in uneven ways.

A different approach, which is the one taken in our work, is to apply a resource assignment algorithm that minimizes, but does not eliminate, the overlapping of the tasks assigned to the same processor. In this context, the tasks' time constraints are not hard, but they are soft and their violations have to be minimized. We will refer to this interval scheduling with soft time constraints problem, as the *Soft Interval Scheduling* problem. The decisions taken are not related to accepting or rejecting a task, since all tasks can be accepted, but on where to assign a task so as to minimize their time overlapping. Then, the overlapping tasks are time shifted as needed to obtain a feasible assignment. Of course, this may cause cascading effects, leading to more time overlaps, implying further deviations between the requested and the actual time intervals. Our objective is to minimize the time shifts required to serve all the requests. This approach acknowledges the fact that such time shifts are actually quite expected, acceptable, and often necessary, since resources are limited and tasks do not generally have deterministic durations.

We can extend the traditional interval scheduling algorithms to be implemented under a soft constraint framework. For example, in the Maxima IS case (see [33]), we can allow tasks' shifts, when a conflict takes place, assigning them to the resource that provides the minimum completion time. In contrast, the conventional approach removes the conflicting task, which is then scheduled in a subsequent scheduling period.

Although the minimization of the time overlaps meets to the degree possible the tasks' QoS requirements, leading to minimal time shifting of the tasks, an additional consideration should be to also optimize resource utilization, so as to obtain a nonwasteful distributed computing architecture. A drawback of existing scheduling approaches is that task assignment is performed either in the direction of maximizing the overall resource utilization efficiency or in the direction of satisfying users' QoS requirements. We argue that a successful algorithm should take into account both considerations, and we address the problem by proposing a novel algorithm that, for a given number of resources, assigns tasks to processors so that 1) *the time overlapping between tasks assigned on the same processor is minimized* (users' QoS requirements are met to the degree possible), while simultaneously 2) *overall resource utilization efficiency is maximized*.

To solve the aforementioned dual-objective task assignment problem, graph partitioning techniques are exploited in this paper. The soft interval scheduling problem is first reduced to a graph partitioning problem, which is then addressed using an M-way graph partitioning methodology [46]. In our case, graph partitioning is performed based on spectral clustering through the use of normalized cuts [47]. The normalized cuts method has advantages over the traditional min-cut methods, as it does not favor the creation of small clusters. To handle this issue, traditional graph partitioning methods, like geometric and multilevel graph partitioning, impose balancing criteria [47]. However, predefining the partition size may be in conflict with the actual data statistics (in our case, the requested start and finish times), where sometimes unbalanced partitions are more appropriate. The use of normalized cuts resolves this issue, by automatically estimating the partition size with respect to the diversity of the tasks' duration within each partition.

We refer to our proposed task allocation scheme as the *Spectral Clustering Scheduling* (abbreviated *SCS*) algorithm. An advantage of the SCS algorithm is that it can be used in two different ways. The first is to efficiently schedule tasks on a given number of resources so as to minimize the task time requirement violations. The second is to find the number of resources or virtual machines required to serve the tasks with a given level of task time violations (or with no violations). The latter approach is particularly helpful in a cloud computing environment, where a resource provider should be able to estimate the service-level agreement (SLA) offered to its users for a given number of virtual machines (resources) used.

The present work extends our earlier work in [48], in several ways. First, we treat the QoS requirements as soft constraints, the violation of which should be minimized, instead of hard constraints (reject overlapping tasks). Second, we define metrics for measuring scheduling efficiency, suitable for the soft constraints case, instead of measuring the performance of the clustering schemes as [48] does. Third, we provide extensive comparisons with other scheduling algorithms (that assume soft and hard time constraints) and we use real-world workload traces to evaluate the efficiency of the proposed method.

2 Problem Formulation

We assume that resource assignment is activated at periodic time intervals of duration T. Within time T, N tasks will be collected and request to be served on one of the M available resources. The users' scheduling needs are expressed through the requested start and finish times of the submitted tasks $T_i, i = 1, 2, \ldots, N$, denoted by ST_i and FT_i, respectively. In what follows, we discuss two different computing paradigms our scheme applies to.

Cloud Computing Paradigm. The proposed method is in good accordance with the cloud Computing paradigm, where users pay for the resource usage with a specific number of computation units (e.g., Amazon EC2). In this scenario, a user requests the time period he/she want to use the cloud resources, expressed as requested start and finish times, ST_i and FT_i. Then, the resource provider activates a set of virtual machines for serving the tasks. Since the activation of a virtual machine entails a cost for the resource provider, an optimization strategy able, on the one hand, to maximize the resources utilization and, on the other, minimize the task overlaps and the consequent time shifts (increase users' satisfaction) are of primary importance for a cloud computing provider.

Grid Computing Paradigm. In this scenario, each task requests service from the grid platform with a specific start and finish time. The problem is to assign each task to a resource so as to obtain 1) minimum violation between the requested and the actual start/finish times achieved, and simultaneously, 2) maximum utilization of the available resources. We will see that these two objectives are simultaneously achieved, for a given number of resources. There is, however, a tradeoff between and the resource utilization and the violation of the tasks' time requirements, obtained by varying the number of resources. In particular, as the number of resources M increases, the time deviation of the tasks decreases and the resource utilization also decreases. Overall, the proposed algorithm optimizes the deviation of the tasks' time requirements at any given utilization value.

The start and finish times can be obtained by the tasks' start time, the respective expected duration, and the tasks' dependencies. If the actual execution time of a task turns out to be larger than the duration initially requested, there are two possibilities for the server. In the first option, the task stops its execution and requests a new resource assignment for the remaining workload (preemption policy). In the second option, the task continues its execution, and thus, the respective resource is unavailable for other tasks (which are then delayed in time, possibly delaying other tasks also) for the corresponding duration. Both scenarios can be addressed by the resource allocation method to be proposed, even though in the current paper, we focus on the first option.

Dependencies among tasks are modeled using a Directed Acyclic Graph $G = \{N, A\}$, where a node $u \in N$ represents a task, and an arc $u \rightarrow v \in A$ represents the dependence of task v on u; v cannot be executed before u finishes execution. Assuming constant communication delays, we can derive the requested start and finish times of all tasks [49] before assigning them to resources.

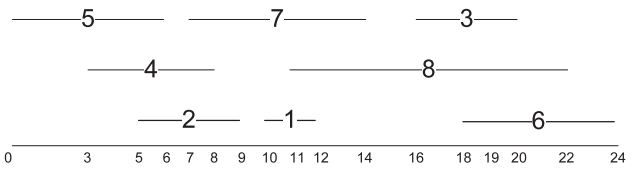


Fig. 1. An example of eight tasks requesting service during a time interval T, and their requested start and finish times.

2.1 Problem Statement

As already mentioned, we set dual objectives for the resource assignment strategy: we want, on the one hand, to minimize the tasks' QoS conflicts and violations and, on the other, to maximize resource utilization. The first objective can be expressed as a minimization of the task overlapping (or equivalently maximization of the task nonoverlapping) among all tasks assigned on the *same* resource, while the second can be expressed as the maximization of the overlapping (nonoverlapping minimization) among all tasks assigned to *different* resources.

In particular, let $V = \{T_1, T_2, \ldots, T_N\}$ be the set of N tasks requesting service, and $R = \{1, 2, \ldots, M\}$ be the set of the M available resources. We also let $T_i \in V \rightarrow r \in R$ be the operator that assigns the task T_i to a resource r. We also denote by C_r the set of tasks assigned to resource $r \in R$, that is, $C_r = \{T_i \in V : T_i \rightarrow r\}$. Then, the optimal resource assignment can be formulated as the problem of finding the optimal selection of the sets $\hat{C}_r$, for all $r \in R$, that simultaneously maximize and minimize the following:

$$\hat{C}_r : \max \sum_{r=1}^{M} \sum_{i \in C_r, j \in C_r} \sigma_{ij,} \min \sum_{r=1}^{M} \sum_{i \in C_r, j \notin C_r} \sigma_{ij}. \quad (1)$$

In (1), σ_{ij} defines the *nonoverlapping degree* between two tasks T_i and T_j

$$\sigma_{ij} = \begin{cases} a, & \text{if } \mathrm{T_i, T_j} \text{ non} - \text{overlapping in time} \\ a(1 - w_{ij}) & \text{if } \mathrm{T_i, T_j} \text{ overlapping in time} \end{cases} \quad (2)$$

where a is a constant and $w_{ij} \in [01]$ represents the percentage of the overlapping duration, defined as the ratio of the time overlapping between T_i and T_j over their overall duration. For the example of Fig. 1, we have $w_{5,4} = 3/(6+5)$ because the duration of 4 and 5 are 6 and 5 units, respectively. It should be noted that the proposed algorithm is independent from the definition of the overlapping measure and that other metrics can also be used.

2.2 Task Order in Using a Resource

In our soft interval scheduling problem, we want to decide the resource and time interval each task will be assigned at to optimize the dual objectives mentioned above. Once the tasks have been assigned to processors, we also have to decide the way time conflicts are resolved. In our approach, the order in which conflicting tasks use a resource is determined from their start time; tasks with the earliest assigned start time are served first.

To indicate the advantages of our soft interval scheduling scheme over the conventional (hard) interval scheduling algorithms, consider the example of Fig. 1, where eight tasks request service within time interval T. Table 1 presents the resource assignment results and the respective time deviation using the (hard) Maxima IS algorithm [37] and the proposed (soft) SCS approach. The maxima IS algorithm initially sorts the tasks in chronological order of their requested start times and then, in case of task overlapping, it removes the conflicting tasks and in particular the one with the largest ending time, and schedules them in the next assignment period.

TABLE 1
The Time Deviations for the Eight Tasks of Fig. 1 in Case of the Maxima IS and the SCS Algorithm

Tasks ID	Resource Assignment Maxima IS	Resource Assignment SCS	Time Deviation Maxima IS	Time Deviation SCS
1	Recourse #2	Recourse #1	0	0
2	Rejected	Recourse #1	19	1
3	Resource #1	Recourse #2	0	0
4	Resource #2	Recourse #2	0	0
5	Resource #1	Recourse #1	0	0
6	Resource #2	Recourse #1	0	2
7	Resource #1	Recourse #2	0	1
8	Rejected	Resource #1	13	1
Total			**32**	**5**
Resource Utilization Maxima IS		Resource Utilization #1: 17/24 Resource Utilization #2: 13/24 **Total Utilization: 30/48=0.625**		
Resource Utilization SCS		Resource Utilization #1: 23/24 Resource Utilization #2: 16/24		

By time deviation, we mean the difference between the requested and the actually assigned start time of a task. As is observed, the accept/reject policy used by conventional interval scheduling approaches is not a fair policy because some tasks undergo much greater time deviations than other tasks. In addition, it is not an optimal strategy in terms of task overlapping and resource utilization.

In contrast, a task assignment policy that tries to minimize task overlapping (as the SCS algorithm, to be described in Section 2, does) instead of rejecting tasks can provide much better results in terms of time deviation and utilization of the resources. For example, consider the case where tasks 1, 2, 5, 6, and 8 are assigned to resource #1, while tasks 3, 4, and 7 are assigned to resource #2. To resolve time overlapping, the task with the smaller requested start time is executed first. Table 1 presents the respective time deviations. It is clear that this assignment is much more fairer than the one obtained by the Maxima IS algorithm, resulting in minimal task overlapping and corresponding time deviations (the total time deviation for SCS is 5 time units while for the Maxima IS it is 32 units). In addition, this policy results in a better exploitation of the resource (62.5 percent for Maxima IS and 81.25 percent for SCS).

2.3 Online Strategy

As already mentioned, despite the minimization of task overlapping [see (1)], it is possible for tasks with overlapping time requirements to be assigned to the same resource. This implies that some of the tasks will be shifted in time and the actual start and finish times of a task, $ST_i^{(a)}$ and $FT_i^{(a)}$, will differ from the requested ones ST_i and FT_i. This dynamically modifies the number of available resources at a given assignment interval.

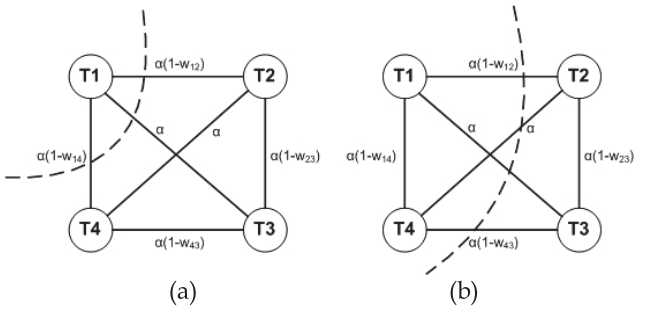


Fig. 2. Interval graph corresponding to four tasks: (a) applying the min-cut criterion, (b) applying the normalized min-cut criterion. Since there are two resources, the graph is partitioned in two parts.

In particular, assume that the resource assignment strategy has been activated k times, and we are at the kth assignment interval $[(k-1)T, kT)$. At this time, the resource assignment policy has already been applied for all tasks whose requested start times are less than $(k-1)T$. Tasks whose requested start times are smaller than kT but greater than $(k-1)T$ are assigned in the current resource assignment interval. If the actual finish time $FT_i^{(a)}$ of an assigned task i (with requested start time smaller than $(k-1)T$) has not elapsed at the kth activation of the resource assignment, task i continues to execute on the same resource in the kth interval, making the resource unavailable until it is completed, and possibly causing other tasks in the kth interval to be shifted. An alternative approach for tasks that are not able to finish execution during their assigned time interval (either due to their being time shifted by other tasks, or because their actual execution time took longer than expected), would be to adopt a preemption policy. In this case, the task stops execution, an overhead is kept by the resource provider and the task requests for a new execution at the next time interval.

3 Joint Optimization of Resource Performance and QoS Requirement

3.1 Graph-Based Representation

A convenient way to solve (1) is to view task assignment as a graph partitioning problem. In particular, let $G = \{V, E\}$ be a weighted undirected graph, the *interval graph*, whose vertices $V = \{T_1, T_2, \ldots, T_N\}$ correspond to the N tasks requesting service within the scheduling period. The weight of an edge between T_i and T_j is taken equal to the nonoverlapping degree $\sigma_{ij} \geq 0$ of the tasks. Since $\sigma_{ij} = \sigma_{ji}$, G is undirected. Note that a graph edge indicates pairwise (dis) similarity between the tasks.

Fig. 2 presents an example of an interval graph with four tasks (nodes), and edges whose weights correspond to the nonoverlapping degrees σ_{ij}. We observe that tasks T_1, T_3 and T_2, T_4 do not overlap [see (2)].

3.2 Normalized Cut Spectral Graph Partitioning

In our approach, a normalized cut spectral graph partitioning algorithm is adopted for resource allocation. This choice was made because the traditional spectral graph partitioning min-cut algorithm [47] would favor the creation of too many small partitions. By optimizing the minimum cut criterion (Fig. 2a), we attempt to assign highly overlapping tasks to different resources (recall the definition of the σ_{ij}s). However, by doing so, we account only for the overlapping of tasks on the same resource and favor the cutting of small sets of isolated nodes in the graph. In Fig. 2a, the minimum cut criterion would assign task T_1 to the first resource, and the other three tasks to the second; something that is undesirable when the overlapping among tasks T_2, T_3, and T_4 is high. On the other hand, by optimizing the normalized minimum cut criterion, the partitioning of the tasks is improved, as seen in Fig. 2b, where the objectives of maximizing the overlapping of tasks in different resources, while minimizing the overlapping in the same resource, are satisfied.

Thus, to avoid solutions that use too many cuts (i.e., too many processors, in the scheduling context), normalization factors are added to (1) [47], as follows:

$$Q_r = \frac{\sum_{i \in C_r, j \in C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}, \qquad \Phi_r = \frac{\sum_{i \in C_r, j \notin C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}. \tag{3}$$

The denominator in (3) is the overlapping degrees of the tasks mapped to resource r for all the N tasks, including the ones assigned to r, and it is used for normalization purposes. Otherwise, optimizing only the numerator of (3) would favor the solution of one task per resource (see Fig. 2). Parameter Q_r expresses a measure of the overall QoS violation for the tasks assigned to the rth processor. Similarly, Φ_r is a measure of the utilization efficiency achieved for processor r. Considering all the M resources of the grid, we define measures Q and Φ for the total tasks' QoS violation and resource utilization as

$$Q = \sum_{r=1}^{M} Q_r \; \Phi = \sum_{r=1}^{M} \Phi_r. \tag{4}$$

3.2.1 Normalized Cut Problem Statement

The formulation of (1) can be rewritten as follows:

$$\hat{C}_r : \max \sum_{r=1}^{M} \frac{\sum_{i \in C_r, j \in C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}, \qquad \min \sum_{r=1}^{M} \frac{\sum_{i \in C_r, j \notin C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}, \tag{5}$$

where $\hat{C}_r$ denotes the set of tasks assigned to resource r, in the optimal solution. Equation (5) ensures that tasks are assigned so as to not only maximize the nonoverlapping degree of all tasks within a resource but also to minimize the nonoverlapping degree among the M available resources. It can be proven, however, that Φ and Q are related through

$$\Phi + Q = M. \tag{6}$$

Given (6), the optimization of (5) can be rewritten as

$$\hat{C}_r : \max \left(M - \sum_{r=1}^{M} \frac{\sum_{i \in C_r, j \notin C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}} \right), \; \min \sum_{r=1}^{M} \frac{\sum_{i \in C_r, j \notin C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}$$

$$\text{or equivalently } \hat{C}_r : \min \sum_{r=1}^{M} \frac{\sum_{i \in C_r, j \notin C_r} \sigma_{ij}}{\sum_{i \in C_r, j \in V} \sigma_{ij}}. \tag{7}$$

Equation (7) shows that the maximization of Q results in the simultaneous minimization of Φ, and vice versa, for a given number of resources. This is intuitively satisfying, since scheduling a set of tasks in a way that makes efficient use of resources is also expected to help meet the QoS requirements of the tasks that are scheduled. Therefore, it is enough to optimize only one of the two criteria [see (6)]. It is

clear, however, that there is a tradeoff that can be obtained by varying the number of resources M, between the resource utilization and the violation of the tasks' time requirements. In particular, as the number of resources increases, the time deviation of the tasks decreases and the resource utilization also decreases.

4 The Scheduling Algorithm

4.1 Matrix Representation

Let us denote by $\mathbf{\Sigma} = [\sigma_{ij}]$ the matrix containing the nonoverlapping measures σ_{ij} for all $N \times N$ pairs of tasks T_i and T_j. Let us also denote by $\mathbf{e}_r = [\cdots e_r^u \cdots]^T$ a $N \times 1$ indicator vector, whose uth entry is given by

$$e_r^u = \begin{cases} 1, & \text{if task } T_u \text{ is assigned to resource } r \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Vector $\mathbf{e}_r$ indicates which tasks are executed on resource r; indices of tasks assigned to resource r are marked with one, and the remaining indices with zero. Since the infrastructure consists of M resources, M different vectors $\mathbf{e}_r$, $r = 1, 2, \ldots, M$, are defined, each specifying the tasks assigned for execution on a given resource. Therefore, the optimization problem of (7) is equivalent to finding the optimal indicator vectors $\hat{\mathbf{e}}_r$.

A difficulty in optimizing (7) is that its right-hand side is *not* express d in terms of the indicator vectors $\mathbf{e}_r$. Thus, the right part of (7) must be rewritten so as to include the vectors $\mathbf{e}_r$. To do so, we denote by $\mathbf{L} = diag(\cdots l_i \cdots)$ the diagonal matrix, whose elements $l_i, i = 1, 2, \ldots, N$, are equal to the cumulative nonoverlapping degrees of T_i with the remaining tasks, that is, $l_i = \sum_{j \in V} \sigma_{ij}$.

Using matrices $\mathbf{L}$ and $\mathbf{\Sigma}$, we express the numerator of (7) as a function of the vectors $\mathbf{e}_r$. In particular, we have

$$\mathbf{e}_r^T(\mathbf{L} - \mathbf{\Sigma})\mathbf{e}_r = \sum_{i \in C_r, j \notin C_r} \sigma_{ij}. \tag{9}$$

In a similar way, the denominator in (9) is written in terms of the indicator vector $\mathbf{e}_r$ as

$$\mathbf{e}_r^T \mathbf{L} \mathbf{e}_r = \sum_{i \in C_r, j \in V} \sigma_{ij}. \tag{10}$$

Using (9) and (10), we can rewrite (7) as

$$\hat{\mathbf{e}}_r, \forall r : \min Q = \min \sum_{r=1}^{M} \frac{\mathbf{e}_r^T(\mathbf{L} - \mathbf{\Sigma})\mathbf{e}_r}{\mathbf{e}_r^T \mathbf{L} \mathbf{e}_r}. \tag{11}$$

4.2 Optimization in the Continuous Domain

Since each task has to be assigned to one resource, the indicator vectors $\mathbf{e}_r$ must take binary values. Thus, if we form the indicator matrix $\mathbf{E} = [\mathbf{e}_1 \cdots \mathbf{e}_M]$, the columns of which correspond to the M resources in the grid and the rows to the N tasks, the rows of $\mathbf{E}$ have only one unit entry and the remaining entries are zero.

One way to solve (11) is to perform the minimization by relaxing the integer constraints, that is, to allow matrix $\mathbf{E}$ to take values in the continuous domain. We denote by $\mathbf{E}_R$ the relaxed version of the indicator matrix $\mathbf{E}$, whose entries take real instead of binary values. The idea is to first find the optimal choice of the relaxed matrix $\mathbf{E}_R$, and then discretize somehow the real values to obtain an approximately optimal integer solution $\mathbf{E}$.

The right part of (11) can be rewritten [51] as

$$Q = M - trace\big(\mathbf{Y}^T \mathbf{L}^{-1/2} \mathbf{\Sigma} \mathbf{L}^{-1/2} \mathbf{Y}\big), \tag{12a}$$

subject to

$$\mathbf{Y}^T \mathbf{Y} = \mathbf{I}, \tag{12b}$$

where matrix $\mathbf{Y}$ is related to matrix $\mathbf{E}_R$ through equation $\mathbf{L}^{-1/2}\mathbf{Y} = \mathbf{E}_R \mathbf{\Lambda}$. $\mathbf{\Lambda}$ is any arbitrary $M \times M$ matrix. In this paper, we select $\mathbf{\Lambda} = \mathbf{I}$. Then, the relaxed matrix $\mathbf{E}_R$, which is actually the matrix we are looking for, is given by

$$\mathbf{E}_R = \mathbf{L}^{-1/2}\mathbf{Y}. \tag{13}$$

Minimization of (12) is obtained through the Ky-Fan theorem [50], which states that the maximum value of $trace(\mathbf{Y}^T \mathbf{L}^{-1/2} \mathbf{\Sigma} \mathbf{L}^{-1/2} \mathbf{Y})$ with respect to matrix $\mathbf{Y}$, subject to the constraint $\mathbf{Y}^T\mathbf{Y} = \mathbf{I}$ is given by the sum of the M ($M < N$) *largest eigenvalues* of matrix $\mathbf{L}^{-1/2} \mathbf{\Sigma} \mathbf{L}^{-1/2}$. Thus,

$$\max_{\text{subject to } \mathbf{Y}^T\mathbf{Y} = \mathbf{I}} \big\{ trace\big(\mathbf{Y}^T \mathbf{L}^{1/2} \mathbf{\Sigma}\, \mathbf{L}^{-1/2} \mathbf{Y}\big) \big\} = \sum_{i=1}^{M} \lambda_i, \tag{14}$$

where λ_i refers to the ith largest eigenvalue of matrix $\mathbf{L}^{-1/2} \mathbf{\Sigma}\, \mathbf{L}^{-1/2}$.

However, the maximization of (14) leads to the minimization of Q in (12a), and the minimum value of Q is

$$\min Q = M - \sum_{i=1}^{M} \lambda_i. \tag{15}$$

The Ky-Fan theorem also states that this minimum value of Q is obtained for the matrix

$$\mathbf{Y} = \mathbf{U}\mathbf{R}, \tag{16}$$

where $\mathbf{U}$ is a $N \times M$ matrix whose columns are the *eigenvectors* corresponding to the M largest eigenvalues of matrix $\mathbf{L}^{-1/2} \mathbf{\Sigma} \mathbf{L}^{-1/2}$ and $\mathbf{R}$ is an *arbitrary rotation matrix* (i.e., orthogonal with determinant of one). Again, a simple approach is to select $\mathbf{R} = \mathbf{I}$, in which case $\mathbf{Y} = \mathbf{U}$.

Therefore, (12) is minimized at $\mathbf{Y} = \mathbf{U}$ and the minimum value is given by (15). The optimal choice $\hat{\mathbf{E}}_R$ of the relaxed matrix $\mathbf{E}_R$ in the continuous domain is

$$\hat{\mathbf{E}}_R = \mathbf{L}^{-1/2}\mathbf{U}. \tag{17}$$

Equation (17) means that the optimal relaxed matrix $\hat{\mathbf{E}}_R$ is related to 1) the cumulative nonoverlapping degree of all tasks and 2) the eigenvectors corresponding to the M largest eigenvalues of matrix $\mathbf{L}^{-1/2} \mathbf{\Sigma} \mathbf{L}^{-1/2}$.

4.3 Rounding the Solution

The optimal matrix $\hat{\mathbf{E}}_R$, given by (17), does not have the form of the indicator matrix $\mathbf{E}$, since the entries of $\hat{\mathbf{E}}_R$ are noninteger, while $\mathbf{E}$'s entries are binary. Consequently, the problem is how to round the continuous values of $\hat{\mathbf{E}}_R$ in a way that approximates matrix $\mathbf{E}$.

A simple rounding process is to set the maximum value of each row of $\hat{\mathbf{E}}_R$ equal to 1 and the remaining values equal to 0. However, this approach yields unsatisfactory performance when there is no dominant maximum value for each row of $\hat{\mathbf{E}}_R$ and it handles the rounding process as N (number of tasks) independent problems.

An alternative approach, which we adopt in this paper, is to treat the N rows of $\hat{\mathbf{E}}_R$ as M-dimensional feature vectors. The algorithm clusters the rows of matrix $\hat{\mathbf{E}}_R$ to M groups (the number of resources). Each row of $\hat{\mathbf{E}}_R$

TABLE 2
The Main Steps of the Proposed Scheduling Algorithm

1. Find the non-overlapping measures σ_{ij} for all tasks, using Eq. (2).	*Initialization*
2. Form the respective matrices **L** and **Σ**.	*Joint Optimization*
3. Compute the eigenvectors of matrix $\mathbf{L}^{-1/2}\mathbf{\Sigma}\,\mathbf{L}^{-1/2}$ Form a matrix **U** of size $N \times M$ whose columns are the *eigenvectors* of the *M* largest eigenvalues of *matrix* $\mathbf{L}^{-1/2}\mathbf{\Sigma}\,\mathbf{L}^{-1/2}$.	*Solution in Continuous Domain*
4. Use Eq. (17) to estimate the continuous matrix $\mathbf{E}_R$ as $\hat{\mathbf{E}}_R = \mathbf{L}^{-1/2}\mathbf{U}$.	
5. Normalize the rows of matrix $\mathbf{E}_R$ so as to lie in the interval [0, 1]. Set iteration=1,	*Solution in Discrete Domain*
a. While (Iteration<MaxIterations) do	
b. Arbitrarily select *M* rows of $\mathbf{E}_R$ as centroids of the *M* groups (processors the tasks are assigned to)	
c. Cluster the remaining rows with respect to the *M* centroids	*k-means*
d. Update the groups centroids as the average of the row entries assigned to a particular group	
e. If the new centroids are different from the old ones go to step c.	
f. Otherwise, Iteration=Iteration+1; go to Step b	

indicates the degree of "fitness" (the association degree) of the corresponding task to each of the M resources. Therefore, the goal of the algorithm is to find the resource to which a task with a specific feature vector fits best.

It has been shown in [51] (see [51, Sections 2.2 and 2.3]) that such an approach, first adopted in [52], provides the minimum Frobenius distance between the continuous and the discrete solution, which is the closest approximate solution to the continuous optimum.

In our case, discretization is achieved via the use of the k-means algorithm on the rows of $\hat{\mathbf{E}}_R$. In particular, we initially normalize the rows of matrix $\hat{\mathbf{E}}_R$ to take values between 0 and 1. Then, we apply the k-means clustering algorithm to these N vectors to form the indicator matrix **E** (for more details see Table 2).

5 Evaluation Metrics

A parameter that affects resource assignment efficiency is the *task granularity g*, defined as the ratio of the average task duration D_u over the time horizon T:

$$g = \frac{D_u}{T}. \tag{18}$$

Task granularity is a measure of how large the generated tasks are compared to the time window T. For example, a value of $g = 0.01$ indicates that the task occupies on the average 1 percent of the scheduling time period for its execution. Another parameter affecting the performance of the algorithms is the number of tasks N that have to be scheduled within the time interval T. Large values of N increases the possibility of tasks' overlapping.

A metric that describes the *load* of the system is

$$B = \frac{ND_u}{T} = Ng, \tag{19}$$

which corresponds to the amount of resources required for the tasks' execution without overlapping in the (unlikely) case that the tasks request consecutive disjoint intervals within the time horizon T. However, in practice, this value is much lower than the actual number of resources needed for serving all tasks with no overlapping. To see that, denote by $\beta(t)$ the number of tasks overlaps at any given time t, and define β as the maximum value of $\beta(t)$ over all t, $0 \le t < T$, that is,

$$\beta = \max_{t \in [0\ T]} \beta(t). \tag{20}$$

Let us denote by $\hat{M}_{opt}$ the minimum number of recourses required to achieve no task overlapping. $\hat{M}_{opt}$ can be found by applying an optimal (exhaustive search) scheduling algorithm. In general, we will use "$\hat{}$" in case where task overlapping is not allowed, and use the same symbol without the "$\hat{}$" when overlapping is allowed. Of course, $\hat{M}_{opt}$ cannot always be found in practice, since the exhaustive search is a nonpolynomial process [47]. Let us also denote by $\hat{M}(A)$ the number of resources required to achieve no task overlapping by some scheduling algorithm A. It is clear that we always have

$$B \le \beta \le \hat{M}_{opt} \le \hat{M}(A). \tag{21}$$

An algorithm A that uses $M(A)$ processors achieves *utilization* of these processors equal to

$$\rho(A) = \frac{B}{M(A)}. \tag{22}$$

It is clear that if $M(A) < \hat{M}_{opt}$, algorithm A fails to feasibly schedule all tasks without overlapping. This can be resolved using two policies; one is to reject the overlapping tasks and assign them to a subsequent scheduling period and the other is to adopt a *"soft scheduling policy"* and remove task overlapping by time shifting the tasks, as discussed in Section 2. In both cases, the QoS performance of algorithm A can be expressed by a *time deviation metric D* (A, ρ) that measures the deviation between the actual and the requested start time for a given utilization ρ:

$$D(A, \rho) = \frac{1}{NT}\sum_{i=1}^{N} \left\| ST_i - ST_i^a \right\|, \tag{23}$$

where ST_i refers to the requested start time of task i, while $ST_i^{(a)}$ to its actual start time when using algorithm A. We expect any reasonable algorithm to satisfy

$$\lim_{\rho \to 0} D(A, \rho) = 0. \tag{24}$$

This is because as utilization tends to zero (by using a large number of processors), the time deviation of any reasonable algorithm should go to zero (no overlapping).

An efficient scheduling algorithm A is not one that minimizes $D(A, \rho)$, potentially using a large number of resources and achieving very low utilization ρ. Instead, it should minimize $D(A, \rho)$ at the highest possible utilization value, i.e., using the minimum number of resources. In case

an algorithm A treats time constraints as hard and has to schedule the tasks without time overlaps, an upper bound on the achievable processor utilization is given by

$$\hat{\rho} = \frac{B}{\hat{M}(A)} \leq \frac{B}{\beta} = \rho_{trace}^{\max} < 1, \quad \text{when } A \text{ uses hard time constraints.} \tag{25}$$

It is clear that $\hat{M}(A) \geq \beta$ because the number of resources estimated by the algorithm A to achieve no task overlapping must be at least greater than the maximum number of tasks' conflicts. Therefore, $\rho_{trace}^{\max} = B/\beta$ is always greater than the utilization achieved by algorithm A [see (25)] resulting in an upper bound with regards to utilization. The upper bound $\rho_{trace}^{\max}$ actually depends on the statistics (trace) of the workload used. Note the upper bound on $\hat{\rho}$ can be much lower than 1. Instead $\rho(A)$ can take values close to 1 at the cost of the increase of the time deviation metric $D(A, \rho)$ due to tasks' overlapping. Therefore, $B/\beta = \rho_{trace}^{\max}$ expresses the maximum utilization achieved assuming a perfect (ideal) scheduling algorithm and it is dependent on granularity and the trace itself because $\rho_{trace}^{\max}$ is affected by the actual tasks' overlaps within a scheduling period T. We use $\rho_{trace}^{\max}$ for comparing the proposed SCS algorithm to the corresponding upper bound. As granularity increases, $\rho_{trace}^{\max}$ increases because smaller actual tasks' overlaps β are encountered within smaller scheduling periods, holding, however, $B \leq \beta$.

Another interesting metric is the ratio of the number of resources utilized by an algorithm over the number of tasks that request to be scheduled:

$$\vartheta = \frac{M(A)}{N}. \tag{26}$$

The inverse of ϑ expresses the operational gain $G = N/M(A)$ for the resource provider:

$$G = \frac{\rho(A)}{g}. \tag{27}$$

Another important parameter is the selection of the scheduling period T. Assuming that the provider dedicates $M(A)$ resources over a time period T, we can define the *computational power* of the provider as

$$P = \frac{M(A)}{T}. \tag{28}$$

For a given number of allocated recourses, the provider's computational power increases as scheduling period T decreases. Then, the following proposition holds:

Proposition 1. *When a resource provider operates under a constant gain G, and assuming that the tasks arrive at a constant rate $\lambda = N/T$, the computational power P of the provider also remains constant.*

The proof of this statement is straightforward since $G = N/M(A) = (N/T)/(M(A)/T) = \lambda/P = constant$.

Proposition 1 indicates that keeping the gain G constant is equivalent to retaining the same computational power for the provider. Thus, for the same traffic statistics (λ and D_u), increasing the tasks' granularity is equivalent to reducing the scheduling period T, resulting in an increase of time deviation delay [see (23)]. According to (27), as granularity increases (decrease of T), utilization $\rho(A)$ increases (since $\rho(A) = G \cdot g$), assuming a constant operational gain (or equivalent computational power). Increase of utilization results in an increase of time deviation $D(A, \rho)$, since fewer resources are used [see (22)].

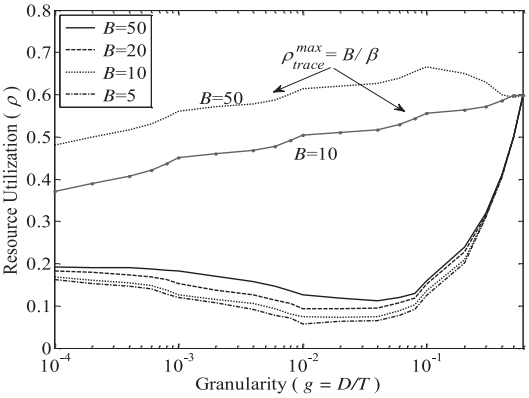


Fig. 3. Utilization factors ρ and upper bound $\rho_{trace}^{\max}$, versus granularity g, for a symmetric and different asymmetric cases.

6 Performance Results

Three different experimental setups were used; synthetically generated tasks through computer simulation, real traces as described in Section 5.1.2, and real-life 3D image rendering tasks, produced in the NEXTGRID project.

6.1 Nonoverlapping Task Scheduling: Zero Time Deviation

In this section, we consider the case where the number of resources is selected in a way that no task overlapping is encountered (in that case, (23) gives $D = 0$).

6.1.1 Probabilistically Generated Tasks

In the first set of experiments, we used a simulator to generate the requested tasks' start and finish times. The simulator allows us to validate the SCS algorithm for different loads B and different scenarios, such as symmetric or asymmetric tasks (in terms of the variance of their duration), and varying degrees of task dependencies. To generate dependent tasks, we let a percentage of the incoming tasks, for example, 20 percent, depend on other tasks, so that they can start their execution only after the completion of the tasks on which they depend.

The tasks' requested start times are generated using a uniform distribution over the time period T. In the case of symmetric tasks, the requested finish time of task i is directly calculated by adding the constant task duration $d_i = d$ to the task start time ST_i; different values for the task durations D are used to assess the SCS algorithm at different granularity values. In the case of asymmetric tasks, we select the tasks' durations randomly and obtain results for different variances of the task durations from their average value. In the simulator, we also force dependencies among the tasks, and vary the degree of dependencies to see the way it affects SCS performance. All the results were obtained by averaging over 500 different realizations (instances) for each given choice of experimental parameters, such as the degree of task dependencies, load B and granularity g.

Fig. 3 presents the processor utilization ρ achieved by the SCS algorithm versus the granularity g, for both symmetric and asymmetric tasks and load $B = 10$. In this

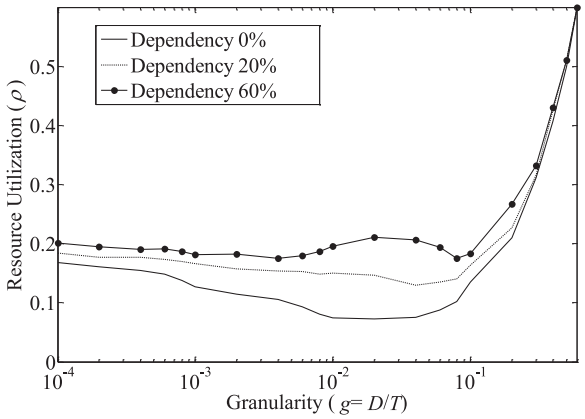


Fig. 4. Utilization ρ versus granularity g for different percentage degree of dependencies.

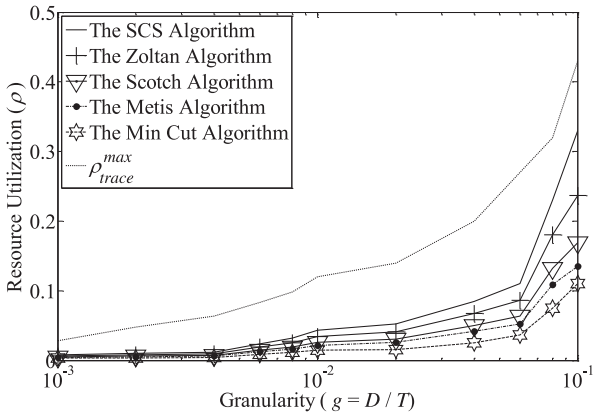


Fig. 5. Utilization ρ versus granularity g for different graph partitioning methods in case of zero time deviation (trace 1). The upper bound $\rho_{trace}^{\max}$ on the utilization is also presented.

TABLE 3
Trace Information That Are Used in the Experiments

	Description
Trace 1	Tasks of various types provided by a resource manager. More details can be found at the report: (http://gwa.ewi.tudelft.nl/pmwiki/reports/gwa-t-1/trace_analysis_report.html)
Trace 2	Experiments from a grid platform consisting of 9 sites geographically located in France (www.grid5000.org).
Trace 3	This log contains accounting records from a large Blue Gene/P system called Intrepid (More details at http://www.cs.huji.ac.il/labs/parallel/workload/l_anl_int/index.html).

figure, we also depict the upper bound $\rho_{trace}^{\max}$. As expected, when asymmetry increases, the processor utilization decreases. The upper bound $\rho_{trace}^{\max}$ cannot be achieved and the difference of the actual utilization from it is about 50 percent for low values of g (fine granularity), increases as g increases, and then drops to zero for high values of g (coarse granularity). For large values of g, almost all tasks overlap, since the task durations are then comparable to the scheduling period duration T, making the minimum number of processors $\hat{M}(A)$ required for no task overlap to be close to the number of tasks N, yielding a utilization close to the upper bound. For small values of g, the number of tasks increases and their duration decreases, slightly improving the performance of SCS over the case of medium granularity.

In Fig. 4, we present results on the performance of the SCS algorithm for varying degrees of dependencies between the tasks. We observe that as the dependencies increase, the utilization factor improves. This is because dependent tasks have a lower degree of overlapping than independent tasks. For example, when all the tasks present dependencies with each other, no task overlapping is encountered, improving resource utilization.

6.1.2 Experiments with Real Trace Logs

In this section, we use real-life workload traces (described in Table 3) for performance evaluation. Initially, we compare the processor utilization achieved by SCS algorithm to that achieved by scheduling algorithms based on other graph partitioning methods, namely, the method proposed by Scotch [53], the method proposed by Metis [54], and the implementation of Zoltan, which is a multilevel hypergraph partitioner [55]. We also compare SCS with the Min-Cuts Tree method [56], which is a bisectional graph partitioning method based on the maximum flow algorithm. This procedure is iteratively applied so as to obtain a k cluster partitioning. *The graph partitioning algorithms operate under the soft constraint paradigm*, as the SCS, and the difference lies in the way each method makes the task to processor assignments.

Fig. 5 presents the processor utilization achieved versus the granularity of the tasks, for the different graph partitioning algorithms. In this figure, we used the real workload logs from the trace #1. As is observed, the proposed SCS algorithm presents better utilization performance than the scheduling schemes that use the other graph partitioning methods examined.

6.2 Overlapping Task Scheduling: Delay Variation versus Resource Utilization Tradeoff

In this section, we evaluate the performance of the SCS strategy for the case where a fixed number of resources is given and thus task overlaps are allowed. As mentioned in Section 5.1, task arrivals in real workload traces present bursts, with many tasks requesting overlapping time intervals, requiring many resources to achieve task non-overlapping (feasible scheduling under the hard constraints), and yielding a low utilization of the resources used. This indicates the importance of the soft interval scheduling concept, introduced in this paper. With soft interval scheduling, we can increase the resources' utilization at the cost of having to resolve task overlaps. Thus, instead of rejecting the overlapping tasks or assigning more resources to them, we shift them in time as needed to obtain a feasible assignment. As the results will indicate the proposed SCS algorithm tends to minimize the time shifts required to serve all the tasks and at the same time maximize the utilization of the resources.

The evaluation of the algorithms is performed using the time deviation measure D [see (23)] versus the resource utilization ρ or the gain G [see (26)]. It is clear from (27) that utilization ρ and gain G are directly related. There is a tradeoff between the time deviation D and the utilization ρ; the higher the time deviation D we can tolerate, the higher the utilization ρ we can achieve. A similar statement holds for the gain G; the higher the gain G is, the higher the time deviation we must tolerate.

It is clear that the best scheduling algorithm is the one that achieves minimum time deviation of the tasks for a

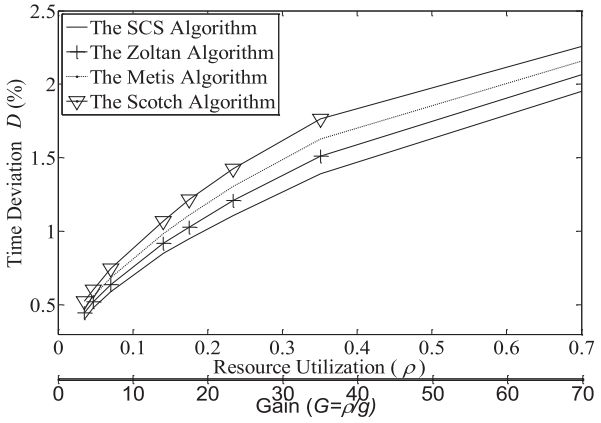


Fig. 6. Time deviation $D(A,\rho)$ for scheduling algorithms based on different graph partitioning methods, as a function of the utilization ρ and *Gain (G)* for the Trace 1.

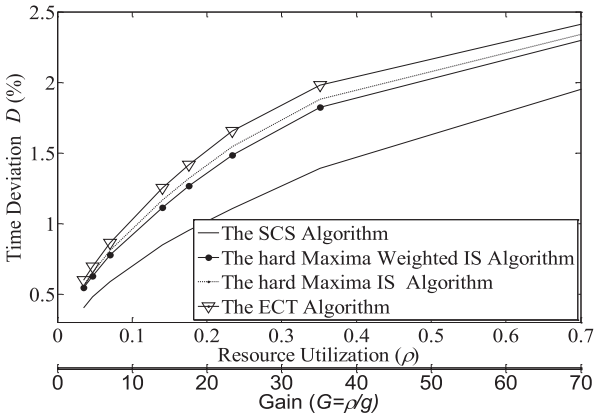


Fig. 7. Time deviation $D(A,\rho)$ for different algorithms A, as a function of the utilization ρ and operational gain (G) , for trace #1.

given utilization of the resources or operational gain G. "Good" scheduling algorithms have D versus ρ curves (or D versus G curves) that lie below the corresponding curves of "bad" algorithms.

6.2.1 Comparisons with Graph Partitioning Algorithms Operate under a "Soft" Constrained Framework

Comparisons of the SCS algorithm are provided initially with scheduling schemes that are based on the previously mentioned graph partitioning algorithms and second with other interval scheduling strategies. The graph partitioning algorithms operate under a "soft constraint" framework, by selecting the resource each task should be executed on, and then, in case of overlapping, shifting the tasks as needed to achieve a feasible solution. The order of execution within a resource is determined by the tasks' start times.

Fig. 6 depicts the time deviation D versus the utilization ρ, for the SCS and other soft-constraints scheduling algorithms that use graph partitioning methods for performing clustering. In the same figure, we have depicted the time deviation D versus the operational gain G, for granularity $g = 0.01$. The best performance is obtained by SCS, since for a given utilization ρ (or operational gain G), it yields the minimum time deviation for the tasks, or, equivalently, for a given time deviation that can be tolerated, it achieves the best processor utilization (uses the fewest processors) or the best ratio for the number of tasks served over the number of resources used.

6.2.2 Comparisons with Hard Constrained Interval Scheduling Algorithms

Regarding the interval scheduling methods, we first examine the Maxima Interval Scheduling algorithm [37]. In this algorithm, the tasks are sorted with respect to their requested start times, and are then assigned to an available resource. In case of conflict, the task with the latest finish time is removed and is assigned to the subsequent scheduling period. The second algorithm used in our comparisons is the Maxima weight IS [37] algorithm. In this algorithm, a graph is constructed with vertices the respective tasks and the edge-weights indicate the id of the first nonoverlapping with the respective vertex task [37]. Then, a max-flow algorithm is used to solve the minimal cost flow problem. The third scheduling algorithm considered is the Earliest Completion Time (ECT) method, which assigns each task to the resource that yields the corresponding minimum completion time.

Fig. 7 compares the SCS with other interval scheduling schemes that resolve conflicts by rejecting tasks. In these experiments, we have again used the soft-constraints version of the Maxima IS algorithm. Again, SCS yields the minimum delay, while ECT yields the maximum one, for a given utilization factor achieved.

6.2.3 Comparisons with Practical Interval Scheduling Algorithms with Soft Deadlines

A practical scheduling algorithm consists of two phases; an algorithm is first used to determine the order in which tasks are served for assignment to processors ("queuing order" phase) and a policy is then used for task-to-processors assignment ("processor assignment" phase). For the queuing order phase, common algorithms are the Earliest Start Time (EST), the Earliest Deadline First, the Shortest Interval (SI) or the Least Laxity First. Regarding the processor assignment phase, the Earliest Completion Time algorithm is usually employed. Since in our case the tasks' QoS are defined by their request start and finish times, and not by deadlines, we have assumed in the results obtained that a task's deadline equals task's finish time multiplied by 2.

To operate the aforementioned practical scheduling algorithms under a "soft" constraints model, conflicting tasks are not rejected, but are shifted and assigned to the resource that minimizes their completion time. In a similar way, we modify the Maxima IS algorithm toward a soft constrained version; tasks are sorted according to their start times, as in the conventional IS algorithm, and, in case of conflicts, the tasks are shifted and assigned to the resource that minimizes their completion time.

Fig. 8 presents the comparison results against practical scheduling algorithms operating under a soft constrained framework. The same parameters as of the Fig. 6 have been used for comparison. We also observe that the best performance is achieved for the proposed SCS scheduling algorithm. It seems that the soft version of the Maxima IS interval scheduling scheme, where the tasks are sorting by the respective start time and then assign using a minimum completion time policy, outperforms the other compared scheduling policies.

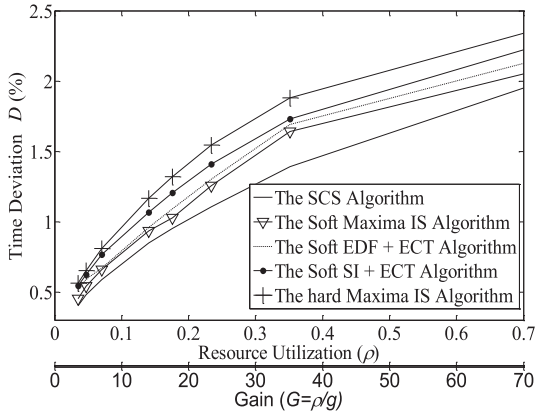


Fig. 8. Time deviation $D(A,\ \rho)$ for different practical scheduling algorithms operating under a "soft" deadline framework as a function of the utilization ρ and operational gain (G), for the trace #1.

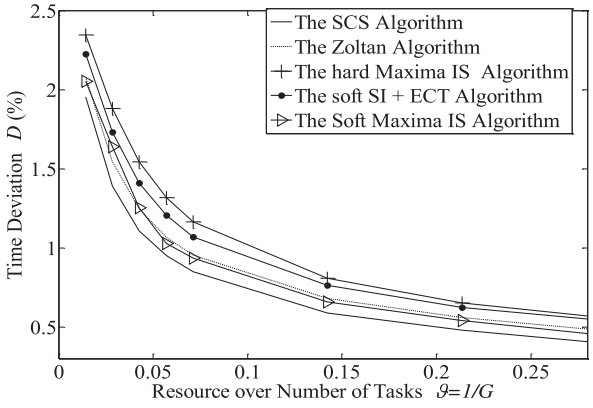


Fig. 9. Time deviation $D(A,\ \rho)$ versus parameter ϑ defined as the ratio of the number of resources over the number of tasks (26).

Fig. 9 presents the time deviation D versus ϑ for the SCS algorithm and other soft deadlines algorithms, either belonging to the category of graph partitioning (Zoltan Algorithm) or of practical scheduling policies (Shortest Interval combined with Earliest Completion Time).

6.2.4 The Effect of Granularity

In the following, we examine the effect of the task granularity g on the time deviation D achieved. When SCS is compared against interval scheduling algorithms that reject tasks that fail to meet their requested deadlines, the performance differences essentially come from the rejections that SCS avoids. The SCS algorithm allows serializing multiple tasks on the same resource within a period, while the other algorithms serialize tasks on the same resource by assigning only one to each period. When granularity takes values close to one and beyond, the differences start to diminish. However, for a constant operational gain G or recourse computational power P (see proposition 1), the time deviation delay increases (i.e., deterioration of end-users' needs) with increasing granularity. As granularity tends to one and the resource provider must ensure small task time deviations, operational gain approaches one (see Fig. 12). This is evident in Fig. 10, where we have plotted the time deviation D versus the granularity g, for operational gain $G \approx 3$. We observe that the difference between the compared algorithms start to diminish for large granularities (especially for $g > 1$). However, in such cases delay significantly increases to retain the same operational gain G (constant ratio of resources over number of tasks). Fig. 11 presents similar comparison results of SCS with other scheduling algorithms, namely, Maxima IS and its soft version, and the soft deadline policy of Zoltan.

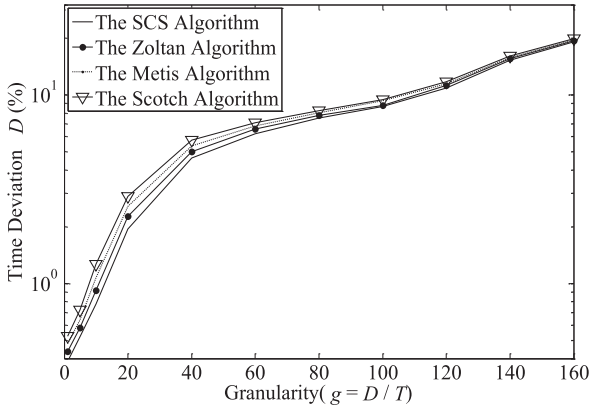


Fig. 10. The effect of granularity g on the time deviation *D(A,p)* for different algorithms operating under the soft-constraints model.

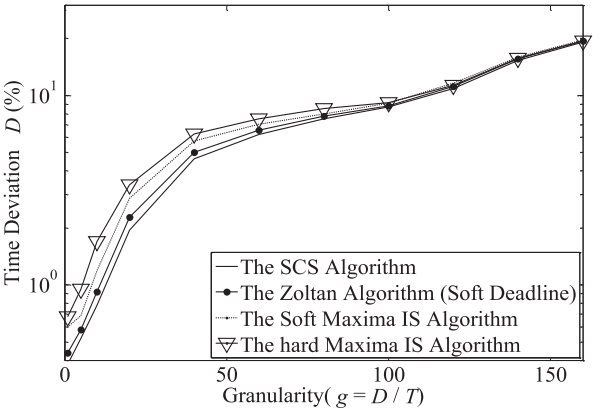


Fig. 11. The effect of granularity g on the time deviation *D(A,p)* for different soft- and hard-constraints algorithms.

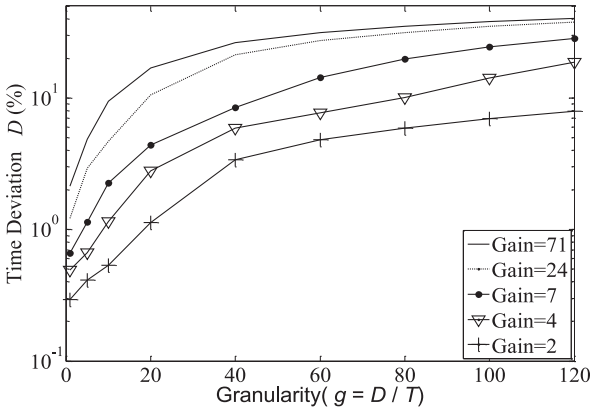


Fig. 12. The effect of the operational gain G on the time deviation *D(A, p)* achieved for a given granularity g.

Instead, if we use a coarse scheduling period for a given operational gain, we achieve small task shifts and better scheduling performance for the end users. Simultaneously, however, utilization decreases meaning that resources are not efficiently used (see (27) and Figs. 6 and 7). Thus, the resource providers can select appropriate scheduling periods that satisfy both the end users and providers' requirements. For example, if the resource provider needs to operate above a certain level utilization level and

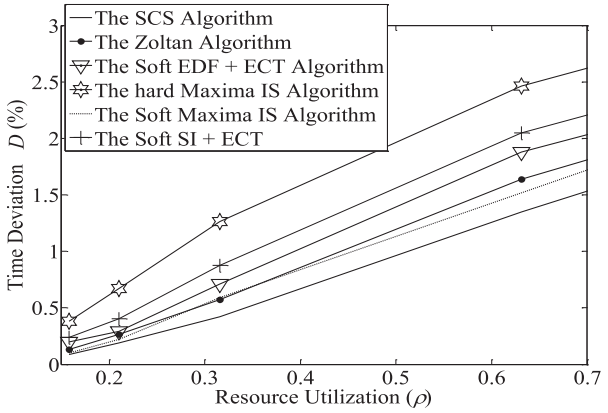


Fig. 13. Time deviation $D(A, \rho)$ for different scheduling algorithms A, as a function of the utilization ρ , for trace #2.

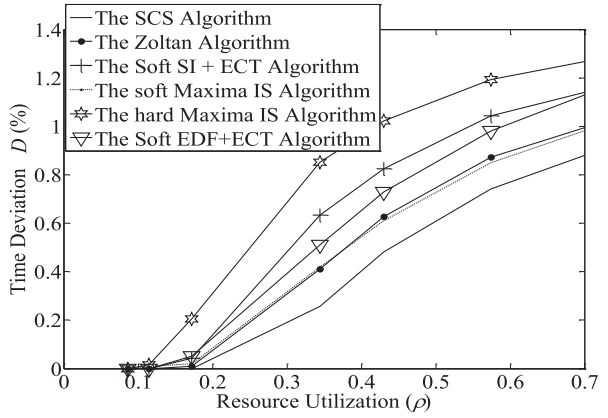


Fig. 14. Time deviation $D(A, \rho)$ for different scheduling algorithms A, as a function of the utilization ρ , for trace #3.

simultaneously giving to end users a time deviation less than a maximum threshold, a balance between granularity and operational gain should be sought using the results of Figs. 6, 7, 8, 9, 10, 11, and 12.

The proposed SCS algorithm yields minimum time deviation for a given utilization (or given operational gain and granularity [see (27)]), compared to all the other examined methods that use either soft or hard constraints. Fig. 12 presents the time deviation versus granularity curve for the SCS algorithm for different values of the operational gain G. The conclusions drawn are similar to those observed in Figs. 10 and 11.

6.2.5 Evaluation over Other Traces

Figs. 13 and 14 present the time deviation D versus utilization ρ curve ($\rho = G \cdot g$) for two additional traces (trace 2 and trace 3). The performance of the scheduling algorithms depends considerably on the trace being used, but for all six traces examined (thee traces of Table 2 plus three additional), SCS exhibited the best performance, since it optimized the time deviation delay for a given utilization ρ or equivalently for a give operational gain G.

Finally, in Fig. 15 we present results obtained using a real-life application scenario, where the generated tasks are 3D image rendering processes submitted for execution in the NEXTGRID grid infrastructure. Each 3D rendering job consists of a series of tasks, each occupying a percentage g (granularity) of *each* time interval T. There exists a maximum duration gT that a task can request in an interval. The choice

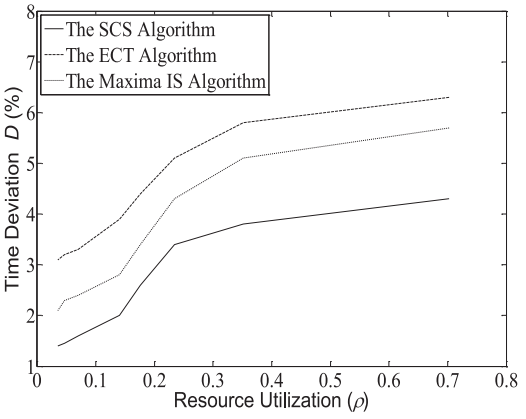


Fig. 15. Time deviation $D(A,p)$ as a function of the utilization p, for 3D rendering tasks generated in the NEXTGRID project.

TABLE 4
Computational Complexity for Different Values of Granularity and Workload Traces

Method	Trace 1			Trace 2			Trace 3		
	Granularity			Granularity			Granularity		
	.01	0.1	0.5	.01	.1	.5	.01	.1	.5
SCS	.7s	.08s	<.01s	2.3s	.3s	.06s	3.7s	.6s	.1s
Zoltan	.4s	.05s	<.01s	1.5s	.09s	.03	2.9s	.3s	.06s
Metis	.1s	.01s	<.01s	1.2s	.07s	<.01s	2.4s	.09s	.03s
Soft Maxima IS	.07s	<.01s	<.01s	.4s	.03s	<.01	2.0s	.04s	<.01s
Maxima IS	.05s	<.01s	<.01s	.3	.01s	<.01	1.9s	.03	<.00s

of the time period T depends on the requested task delay and, thus, on the type of tasks; for interactive tasks (e.g., mouse movement) we need lower delay than application tasks (e.g., 3D rendering). Therefore, we have different schedulers for different types of tasks. The interactive tasks (e.g., mouse movement) present very small execution time, and thus, selection of a large T would result to too low granularity. For this type of task, we select T to be 30 ms. Instead, 3D rending tasks present much higher execution times, and thus, we need to select higher periods, 800 ms.

6.3 Computational Complexity

The SCS scheduling algorithm consists of two main steps: 1) the eigenvalue decomposition optimization algorithm and 2) the clustering algorithm (k-means in our case). The fastest implementation for the eigenvalue decomposition problem is though the Lanczos method whose complexity is $O(MN^2\tau)$, where M is the number of eigenvalues that have to be computed (equal to the number of processors), N is the number of tasks, and τ the number of iterations of the algorithm. In our case, τ takes small values (around 20-40), compared to M and particularly N. However, for low granularity values, M is several times smaller than N, making the complexity to be of order N^2 in that case in practice. The clustering step of the SCS algorithm is implemented using the k-means method, which has complexity $O(N^2\tau)$ in our case.

In Table 4, we present experiments regarding the computational complexity performance of the SCS algorithm, which is compared to that of other scheduling

methods. The experiments have been carried out for three examined traces at different granularities values on an Intel Core2 Duo 3.00 GHz. It is clear that as the granularity increases the execution time of the scheduler decreases, since fewer tasks are encountered. Although SCS presents the worst execution time performance, the actual runtime is not so critical even for low granularities values.

7 Conclusions

The proposed spectral clustering scheduling scheme aims at maximizing processor utilization efficiency, while simultaneously minimizing the tasks' QoS degradation. Task QoS is specified through the requested start and finish times, while QoS degradation is expressed through a time deviation metric D. Resource assignment is viewed as a normalized cuts spectral graph partitioning problem. The algorithm defines a nonoverlapping measure between tasks, and then uses a matrix representation, the notion of generalized eigenvalues, and the Ky-Fan theorem to perform scheduling (graph partitioning) in the relaxed continuous space. The solution of the continuous relaxation is then rounded to a discrete solution.

Experimental results and comparisons of the SCS algorithm with other interval scheduling algorithms (such as the traditional min-cut method, the methods of Scotch, Metis, Zoltan, and the Maxima IS, ECT and Max Flow Interval scheduling) were carried out. We tested the SCS scheme under two models; the nonoverlapping model, where we find the minimum number of processors needed to achieve zero time deviation for all tasks (hard time constraints case) and the model, where the tasks assigned to a processor are allowed to overlap and have to be shifted in time (soft time constraints case), in which case we measure the time deviation encountered as a function of the utilization achieved. In both cases, the SCS algorithm outperforms the other scheduling and graph partitioning algorithms examined. The experiments were carried out with probabilistically generated data and real-life cases of 3D image rendering processes.

Acknowledgments

The authors would like to thank the Advanced School for Computing and Imaging (special thanks to Dr. Henri Bal) for the Trace 1, the Grid'5000 team (special thanks to Dr. Franck Cappello, Dr. Olivier Richard and to Nicolas Capit) for the Trace 2, and the Susan Coghlan, Narayan Desai and Wei Tang regarding Trace 3.

References

[1] C. Castillo, G. Rouskas, and K. Harfoush, "On the Design of Online Scheduling Algorithms for Advance Reservations and QoS in Grids," *Proc. IEEE Int'l Conf. Parallel and Distributed Processing Symp. (PDP),* pp. 1-10, Mar. 2007.

[2] N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou, and E. Varvarigos, "A Combined Fuzzy -Neural Network Model for Non-Linear Prediction of 3D Rendering Workload in Grid Computing," *IEEE Trans. Systems, Man, and Cybernetics (SMC)-Part-B,* vol. 34, no. 2, pp. 1235-1247, Apr. 2004.

[3] E. Arkin and E. Silverberg, "Scheduling Tasks with Fixed Start and End Times," *Discrete Applied Math.,* vol. 18, no. 1, pp. 1-8, 1987.

[4] R.W. Lucky, "Cloud Computing," *IEEE Spectrum,* vol. 46, no. 5, p. 27, May 2009.

[5] K. Singh, E. İpek, S.A. McKee, B.R. de Supinski, M. Schulz, and R. Caruana, "Predicting Parallel Application Performance via Machine Learning Approaches," *Concurrency and Computation: Practice & Experience,* vol. 19, no. 17, pp. 2219-2235, Dec. 2007.

[6] M. Maheswaran, K. Krauter, and R. Buyya, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," *Software: Practice and Experience,* vol. 32, no. 2, pp. 135-164, Feb. 2002.

[7] R.J. Al-Ali et al., "Analysis and Provision of QoS for Distributed Grid Applications," *J. Grid Computing,* vol. 2, pp. 163-182, 2004.

[8] M.S. Fineberg and O. Serlin, "Multiprogramming for Hybrid Computation," *Proc. IFIPS Fall Joint Computer Conf.,* 1967.

[9] A. Stankovic et al., "Implications of Classical Scheduling Results for Real Time Systems," *Computer,* vol. 28, no. 6, pp. 16-25, June 1995.

[10] P. Kokkinos and E. Varvarigos, "A Framework for Providing Hard Delay Guarantees and User Fairness in Grid Computing," *Future Generation Computer Systems,* vol. 25, no. 6, pp. 674-686, 2009.

[11] D. Jackson, Q. Snell, and M. Clement, "Core Algorithms of the Maui Scheduler," *Proc. Seventh Int'l Workshop Job Scheduling Strategies for Parallel Processing (JSSPP),* pp. 87-102, 2001.

[12] B. Bode et al., "The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters," *Proc. Usenix Conf.,* 2000.

[13] "Platform Computing Corporation," http://www.platform.com, 2013.

[14] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," *Proc. Ninth Heterogeneous Computing Workshop,* pp. 349-363, 2000.

[15] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal, "Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimisation Algorithm," *Software: Practice and Experience,* vol. 35, pp. 491-512, 2005.

[16] N. Doulamis, A. Doulamis, E. Varvarigos, and T. Varvarigou, "Fair Scheduling Algorithms in Grids," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 11, pp. 1630-1648, Nov. 2007.

[17] K. Rzadca, D. Trystram, and A. Wierzbicki, "Fair Game-Theoretic Resource Management in Dedicated Grids," *Proc. IEEE Seventh Int'l Symp. Cluster Computing and the Grid (CCGrid),* pp. 343-350, 2007.

[18] V. Martino and M. Mililotti, "Scheduling in a Grid Computing Environment Using Genetic Algorithm," *Proc. 16th Int'l Parallel and Distributed Processing Symp.,* p. 297, Apr. 2002.

[19] S. Kim and J. Weissman, "A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. IEEE Int'l Conf. Parallel Processing (ICPP),* pp. 406-413, Aug. 2004.

[20] H. Yin, H. Wu, and J. Zhou, "An Improved Genetic Algorithm with Limited Iteration for Grid Scheduling," *Proc. IEEE Int'l Conf. Grid and Cooperative Computing (GCC),* pp. 221-227, Aug. 2007.

[21] G. Ye, R. Rao, and M. Li, "A Multiobjective Resources Scheduling Approach Based on Genetic Algorithms in Grid Environment," *Proc. Fifth Int'l Conf. Grid and Cooperative Computing Workshops (GCCW '06),* pp. 504-509, Oct. 2006.

[22] W. Smith, I. Foster, and V. Taylor, "Scheduling with Advanced Reservations," *Proc. 14th Int'l Parallel and Distributed Symp. (IPDPS),* pp. 127-132, 2000.

[23] E. Varvarigos, N. Doulamis, A. Doulamis, and T. Varvarigou, "Timed/Advance Reservation Schemes and Scheduling Algorithms for QoS Resource Management in Grids," *Engineering the Grid,* pp. 355-378, Am. Scientific Publishers, 2006.

[24] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation," *Proc. Seventh Int'l Workshop Quality of Service (IWQOS),* pp. 27-36, 1999.

[25] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail, "G-QoSM: Grid Service Discovery Using QoS Properties," *J. Computing and Informatics,* vol. 21, no. 4, pp. 363-382, 2002.

[26] Y. Zhang et al., "Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling," *Proc. IEEE Conf. Cluster Computing and the Grid,* pp. 568-575, May 2006.

[27] H. Topcuoglu, S. Hariri, and M.Y. Wu, "Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 3, pp. 260-274, Mar. 2002.

[28] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson, "Scheduling Strategies for Mapping Application Workflows Onto the Grid," *Proc. IEEE Symp. High Performance Distributed Computing,* pp. 125-134, 2005.

[29] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," *Proc. IEEE 18th Int'l Parallel and Distributed Processing Symp.,* p. 111, 2004.

[30] W. Kubiak, B. Penz, and D. Trystram, "Scheduling Chains on Uniform Processors with Communication Delays," *J. Scheduling,* vol. 5, no. 6, pp. 459-476, 2002.

[31] T. Yang and A. Gerasoulis, "DSC: Scheduling of Parallel Task on a Unbounded number of Processors," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 5, pp. 951-967, Sept. 1994.

[32] Y. Zhang, C. Koelbel, and K. Kennedy, "Relative Performance of Scheduling Algorithms in Grid Environments," *Proc. IEEE Seventh Int'l Conf. Cluster Computing and the Grid,* pp. 521-528, May 2007.

[33] A. Kolen, J. Lenstra, C. Papadimitriou, and F. Spieksma, "Interval Scheduling: A Survey," *Naval Research Logistics,* vol. 54, no. 5, pp. 530-543, 2007.

[34] R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor, "Algorithmic Aspects of Bandwidth Trading," *ACM Trans. Algorithms,* vol. 3, article 10, 2007.

[35] K. Nakajima and S. Hakimi, "Complexity Results for Scheduling Tasks with Discrete Starting Times," *J. Algorithms,* vol. 3, no. 4, pp. 344-361, 1982.

[36] R. Lipton and A. Tomkins, "Online Interval Scheduling," *Proc. Ann. ACM SIAM Symp. Discrete Algorithms,* pp. 302-311, 1994.

[37] K.I. Bouzina and H. Emmons, "Interval Scheduling on Identical Machines," *J. Global Optimization,* vol. 9, pp. 379-393, 1996.

[38] M.C. Carlisle and E.L. Lloyd, "On the K-Coloring of Intervals," *Discrete Applied Math,* vol. 59, pp. 225-235, 1995.

[39] U. Faigle and W.M. Nawijn, "Note on Scheduling Intervals On-Line," *Discrete Applied Math.,* vol. 58, pp. 13-17, 1995.

[40] L.G. Kroon, M. Salomon, and L. van Wassenhove, "Exact and Approximation Algorithms for the Operational Fixed Interval Scheduling Problem," *European J. Operational Research,* vol. 82, pp. 190-205, 1995.

[41] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, 1980.

[42] H.D. Karatza, "Periodic Task Cluster Scheduling in Distributed Systems," *Computer System Performance Modeling in Perspective, World Scientific,* pp. 257-276, Imperial College Press, 2006.

[43] K. Gkoutioudi and H.D. Karatza, "Task Cluster Scheduling in a Grid System," *Simulation Modelling Practice and Theory,* vol. 18, no. 9, pp. 1242-1252, Oct. 2010.

[44] D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini, and G.R. Nudd, "Local Grid Scheduling Techniques Using Performance Prediction," *IEEE Proc. Computers and Digital Techniques,* vol. 150, no. 2, pp. 87-96, Mar. 2003.

[45] S. Vadhiyar and J. Dongarra, "A Metascheduler for the Grid," *Proc. Int'l Symp. High Performance Distributed Computing,* 2002.

[46] G. Karypis and V. Kumar, "Multilevel K-Way Partitioning Scheme for Irregular Graphs," Technical Report TR 95-064, Dept. of Computer Science, Univ. of Minnesota, 1995.

[47] J. Shi and J. Malik, "Normalized Cut and Image Segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 8, pp. 888-905, Aug. 2000.

[48] N. Doulamis, P. Kokkinos, and E. Varvarigos, "Spectral Clustering Scheduling Techniques for Tasks with Strict QoS Requirements'," *Proc. 14th Int'l Conf. Parallel and Distributed Systems,* 2008.

[49] T.H. Cormen, C.E. Charles, R.L. Rivest, and C. Stein, *Introduction to Algorithms,* second ed., MIT Press and McGraw-Hill, 2001.

[50] K. Fan, "Maximum Properties and Inequalities for the Eigenvalues of Completely Continuous Operators," *Proc. Nat'l Academy of Sciences of USA,* vol. 37, pp. 760-766, 1951.

[51] F.R. Bach and M.I. Jordan, "Learning Spectral Clustering," *Proc. Advances in Neural Information Processing Systems (NIPS),* 2004.

[52] H. Zha, C. Ding, M. Gu, X. He, and H. Simon, "Spectral Relaxation for K-Means Clustering," *Proc. Advances in Neural Information Processing Systems (NIPS),* 2002.

[53] F. Pellegrini and J. Roman, "Scotch: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs," *Proc. Int'l Conf. High-Performance Computing and Networking (HPCN '96),* pp. 493-498, Apr. 1996.

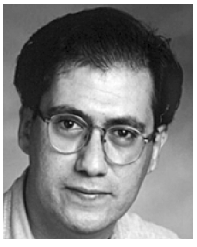
[54] G. Karypis and V. Kumar, "METIS:A Software Package for Partitioning Unstructured Graphs, Partitioning, Meshes, and Computing Fill-Reducing Ordering of Sparse Matrices," technical report, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, 2002.

[55] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, "Parallel Hypergraph Partitioning for Scientific Computing," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* 2006.

[56] Z. Wu and R. Leahy, "An Optimal Graph Theoretic Approach to Data Clustering: Theory and Its Application to Image Segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 11, pp. 1101-1113, Nov. 1993.

Nikolaos D. Doulamis (S'96-M'00) received the Diploma in electrical and computer engineering from the National Technical University of Athens (NTUA) in 1995 with the highest honor and the PhD degree from the same department in 2001. He is currently an assistant professor at NTUA. He received the Best Greek Student award in the field of engineering at national level by the Technical Chamber of Greece in 1995, Best Graduate Thesis Award in the area of electrical engineering, and of NTUA's Best Young Engineer Medal. He has served as the chairman or member of the program committee of more than 40 international conferences. He is an author of more than 45 journals papers in the field of video transmission, content-based image retrieval, and grid computing, and of more than 140 conference papers. Fifteen of the journals papers have been published in the IEEE press. He has more than 1,600 citations. He is a member of the IEEE.

Panagiotis Kokkinos received the Diploma in computer engineering and informatics in 2003 and the MS degree in integrated software and hardware systems in 2006, both from the University of Patras, Greece. He is currently working toward the PhD degree in computer engineering and informatics at the University of Patras. His research activities are in the areas of ad hoc networks and grid computing.



Emmanouel (Manos) Varvarigos received the Diploma in electrical and computer engineering from the National Technical University of Athens in 1988, and the MS and PhD degrees in electrical engineering and computer science from the Massachusetts Institute of Technology in 1990 and 1992, respectively. He has held faculty positions at the University of California, Santa Barbara (1992-1998, as an associate professor) and Delft University of Technology, the Netherlands (1998-2000, as an associate professor). Since 2000 he has been a professor of computer engineering and informatics at the University of Patras, Greece, where he heads the Communication Networks Lab. He is also the director of the Network Technologies Sector at the Computer Technology Institute, which has a major role in the development of network technologies in Greece. He has served as an organizer and program committee of several international conferences, primarily in the networking area. He was also a researcher at Bell Labs, and consulted with companies in the US and in Europe. His research activities are in the areas of protocols for high-speed networks, network services, parallel and distributed computation, ad hoc networks, and grid computing.