Computer Networks 52 (2008) 1762–1781

Contents lists available at ScienceDirect

Computer Networks

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

Self-adjustment of resource allocation for grid applications

Daniel M. Batista[a], Nelson L.S. da Fonseca[a,*], Flavio K. Miyazawa[a], Fabrizio Granelli[b]

[a] *Institute of Computing, State University of Campinas, CxP 6176, Av. Albert Einstein 1251, Campinas, SP 13084-971, Brazil*
[b] *DIT, University of Trento, Via Sommarive 14, I-38050 Trento, Italy*

ARTICLE INFO

Article history:
Received 9 May 2007
Received in revised form 17 December 2007
Accepted 6 March 2008
Available online 13 March 2008

Responsible Editor: N. Akar

Keywords:
Grid networks
Task scheduling
Reactive scheduling
Task migration

ABSTRACT

Grids involve coordinated resource sharing and problem solving in heterogeneous dynamic environments to meet the needs of a generation of researchers requiring large amounts of bandwidth and more powerful computational resources. The lack of resource ownership by grid schedulers and fluctuations in resource availability require mechanisms which will enable grids to adjust themselves to cope with fluctuations. The lack of a central controller implies a need for self-adaptation. Grids must thus be enabled with the ability to discover, monitor and manage the use of resources so they can operate autonomously. Two different approaches have been conceived to match the resource demands of grid applications to resource availability: Dynamic scheduling and adaptive scheduling. However, these two approaches fail to address at least one of three important issues: (i) the production of feasible schedules in a reasonable amount of time in relation to that required for the execution of an application; (ii) the impact of network link availability on the execution time of an application; and (iii) the necessity of migrating codes to decrease the execution time of an application. To overcome these challenges, this paper proposes a procedure for enabling grid applications, composed of various dependent tasks, to deal with the availability of hosts and links bandwidth. This procedure involves task scheduling, resource monitoring and task migration, with the goal of decreasing the execution time of grid applications. The procedure differs from other approaches in the literature because it constantly considers changes in resource availability, especially network bandwidth availability, to trigger task migration. The proposed procedure is illustrated via simulation using various scenarios involving fluctuation of resource availability. An additional contribution of this paper is the introduction of a set of schedulers offering solutions which differ in terms of both schedule length and computational complexity. The distinguishing aspect of this set of schedulers is the consideration of time requirements in the production of feasible schedules. Performance is then evaluated considering various network topologies and task dependencies.

1. Introduction

Grid networks (Grids) have been designed to provide a distributed computational infrastructure for advanced science and engineering [1,2]. They involve coordinated resource sharing and problem solving in heterogeneous dynamic environments to meet the needs of a generation of researchers requiring large amounts of bandwidth and more powerful computational resources. Although in its infancy, cooperative problem solving via grids has become a reality, and various areas from aircraft engineering to bioinformatics have benefited from this novel technology. Grids are expected to evolve from pure research information processing to e-commerce, as has happened with the World Wide Web.

* Corresponding author. Tel.: +55 19 3521 5878; fax: +55 19 3521 5847.
E-mail addresses: batista@ic.unicamp.br (D.M. Batista), nfonseca@ic.unicamp.br (N.L.S. da Fonseca), fkm@ic.unicamp.br (F.K. Miyazawa), granelli@dit.unitn.it (F. Granelli).

1389-1286/$ - see front matter © 2008 Published by Elsevier B.V.
doi:10.1016/j.comnet.2008.03.002

Central to grid processing is the scheduling of application tasks to resources. The lack of resource ownership by grid schedulers and fluctuations in resource availability require mechanisms which will enable grids to adjust themselves to cope with fluctuations. A sudden increase in link load can, for example, increase the time for the transfer of data between the computers where two tasks reside, thus leading to the necessity of relocating the tasks to a third computer. Furthermore, the lack of a central controller implies a need for self-adaptation. The ability to discover, monitor and manage the use of resources is fundamental for the autonomous operation of a grid.

Dynamic scheduling and adaptive scheduling are two different approaches designed to match the resource demands of grid applications to resource availability. Dynamic scheduling [3] is employed when not all the resource requirements of an application are known at the time of the scheduling of the first tasks composing the applications. In a direct acyclic graph (DAG) representation of an application, such a situation is represented by unknown edge and node weights, which prevents the definition of a schedule involving all tasks at the initial scheduling time. These unknown demands are discovered only after the completion of certain tasks, and the taking of decisions about resource allocation to tasks with unknown demands is postponed until the moment in which dependencies are resolved. Thus, the scheduling of tasks is pursued in several steps, providing a certain adaptability to the availability of resource.

Adaptive scheduling [4] is employed to cope with resource availability fluctuations. Resources are monitored by continuous measurement which provides a precise view of their availability at the scheduling time of each task. Adaptive scheduling can be applied to any application whereas dynamic scheduling only to those with unknowns demands.

Although both dynamic scheduling and adaptive scheduling take into consideration the dynamics of resource availability, such availability is verified only at specific instants. Dynamic scheduling verifies this availability only when previously unknown demands are resolved, whereas adaptive scheduling checks the state of the grid only when scheduling a task. These schemes are quite restrictive and fail to exploit various opportunities involving resource availability, in this way, preventing a dynamic search for the minimum execution time of an application. Changes during the execution of a task are neglected, although this can increase the execution time. Furthermore, both approaches fail to address at least one of three important issues: (i) the production of feasible schedules in a reasonable amount of time in relation to that required for the execution of an application; (ii) the impact of network link availability on the execution time of an application; and (iii) the necessity of migrating codes to decrease the execution time of an application.

It is, however, imperative to consider changes in resource availability at all times during the execution of the tasks composing an application. This need has been recognized in previous papers [5–9,4,10,11]. However, all the solutions adopted in an attempt to overcome the problem have failed to address at least one of the following issues: (i) consideration of network performance degradation as a source for triggering task migration; (ii) accountability of overhead for transferring data between tasks; (iii) evaluation of the benefits of task migration considering both overhead involved and the remaining workload to be processed; (iv) availability of recently released resources; (v) consideration of the existing dependencies between tasks; (vi) consideration of deadlines in the production of schedules.

The present paper, however, proposes a novel procedure for enabling grid applications composed of various dependent tasks to meet all these requirements. It is related to the availability of hosts and link bandwidth. This procedure involves task scheduling, resource monitoring and task migration, with the goal of decreasing the execution time of grid applications. The procedure for self-adjustment differs from other approaches in the literature by considering changes in resource availability, especially network bandwidth, the whole time, using this information to evaluate the benefits of changes and trigger task migration. To our knowledge no other proposal address these issues in the way in which they are addressed here. It is especially appropriate for applications composed of dependent tasks with huge demands for data transfer, as are typical of e-Science applications.

Moreover, in our approach the benefits of task migration are always verified against the overhead paid by such migrations, so that a minimum execution time can be achieved. The procedure introduced in this paper is executed by individual applications, which are empowered with autonomy and control designed to minimize execution time. The overall maximization of the utilization of a grid resource is, however, beyond the scope of the proposal.

The scheduling problem is an NP-hard problem, and feasible solutions in real time require either heuristics or approximations. Moreover, computational complexity is increased because the need to account for heterogeneous resources and irregular topologies, which contrasts to what happens in multiprocessor systems. An additional contribution of this paper is the introduction of a set of schedulers offering solutions which differ in terms of both schedule length and computational complexity. The distinguishing aspect of this set of schedulers is the consideration of time requirements in the production of feasible schedules. Performance is then evaluated considering various network topologies and task dependencies.

This paper is organized as following. Section 2 introduces the proposed procedure for self-adjustment. Section 3 introduces eight novel schedulers. Section 4 provides numerical examples. Section 5 discusses related work and Section 6 furnishes some conclusions.

2. Procedure for self-adjustment of resource allocation

Key to the performance of grid applications is the choice of resources composing the virtual organization (computing system) to be used to execute the application. This choice is made by schedulers. Fig. 1 illustrates the various phases in the execution of a grid application, with the bottom left showing the steps needed for scheduling.

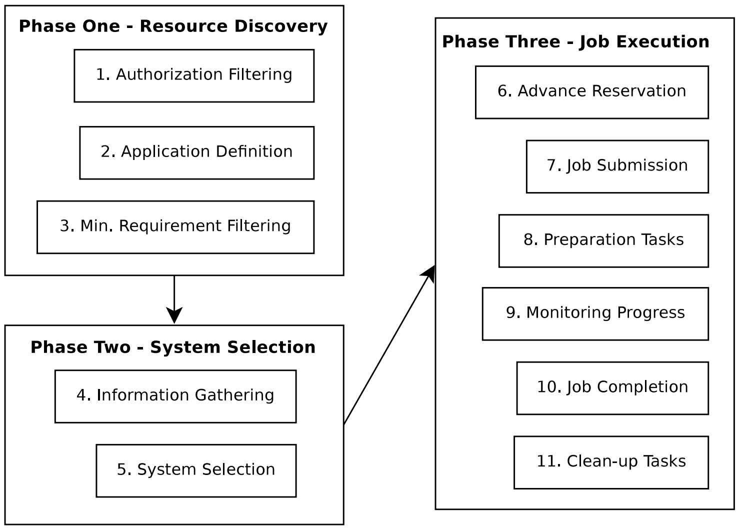
Phase One - Resource Discovery
1. Authorization Filtering
2. Application Definition
3. Min. Requirement Filtering
Phase Two - System Selection
4. Information Gathering
5. System Selection
Phase Three - Job Execution
6. Advance Reservation
7. Job Submission
8. Preparation Tasks
9. Monitoring Progress
10. Job Completion
11. Clean-up Tasks

Fig. 1. Phases of a grid application execution [12].

Resource discovery and determination of application requirements constitutes the first phase of the process. The main issue in scheduling is how to map the tasks of an application onto available resources so that objectives can be achieved. The procedure introduced in this paper aims at minimizing the execution time of the application (schedule length) and considers applications composed of tasks which can be described as direct acyclic graphs (DAGs); in these applications, vertices represent the tasks to be performed and the arcs the dependence between two tasks. The weights of the arcs represent the amount of data to be exchanged by the tasks and the weights of the vertices the amount of processing required for a task. Several e-science applications, such as those in astronomy and the simulation of molecular dynamics, can be represented with DAGs. Fig. 2 illustrates the DAG of a visualization application (remote rendering) [13] that will be used to illustrate the procedure for self-adjustment.

In this paper, grids are represented by a set of hosts connected by network links. CPU and bandwidth demands are considered, although other demands are not taken into account. This limitation does not mean that the approach is limited to the consideration of these demands, but is rather a question involving ease of illustration.

Once tasks are allocated to hosts (grid nodes) according to a schedule, they are executed until all have been completed. However, due to the lack of ownership of resources, availability can change dynamically due to other loads on the grid. Thus, the original schedule may become sub-optimal. If, for instance, the load of a processor decreases, this processor may become an interesting choice for decreasing the execution time of the application. Therefore, if changes in resource availability lead to changes in the predicted schedule length, the schedule must be redefined so that a shorter schedule than that originally predicted will be achieved. Indeed, the procedure for self-adjustment enables grid applications to adapt themselves to current resource availability [14].

Although Step 9 in Fig. 1 can detect performance degradation, the availability of new resources is not considered. In order to provide adaptation to any type of event affecting the availability of resources, it is necessary to monitor networked resources periodically and perform task migration accordingly. Task migration is designed to reduce the time of execution of a single application, rather than the overall optimization of the utilization of grid resources. The benefit of potential migrations is always balanced by the overhead necessary to realize them since the transfer

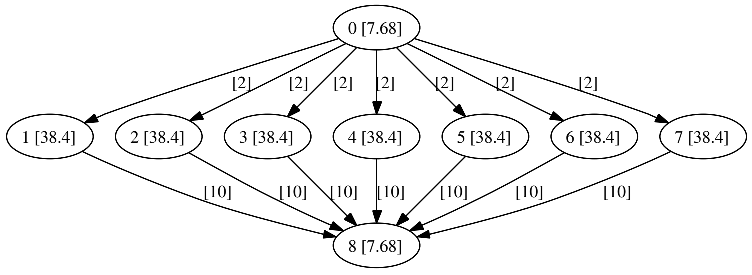

Fig. 2. A grid application DAG.

of code, data and processing context contributes to overhead. The cost of migration must be accounted for in all potential rescheduling of tasks. The accountability of task migration and bandwidth availability for data transfer represent a unique aspect of the proposed procedure. Note that information about resource availability can be shared by all applications of the grid.

The present proposal involves the following steps:

- *Step 1* Map the DAG describing the tasks that represent an application to the graph describing the grid resources. Produce a schedule for the beginning of task execution and data transfer;
- *Step 2* Transfer the task codes and data to the hosts where the tasks will run. The execution of the tasks begins as soon as transfer is completed;
- *Step 3* Monitor the resources of the grid to detect any variation in availability of resources, either decrease or increase;
- *Step 4* Gather the data collected in Step 3 and compare it to the scenario used for previous task scheduling. If no change is detected, continue periodic monitoring of the grid (Step 3);
- *Step 5* Derive a new DAG representing current computational and data transfer demands and produce a schedule for these tasks;
- *Step 6* Check whether the schedule derived is the same as the current one;
- *Step 7* Compare the cost of the solution derived in Step 5 with the cost of the current solution. The cost of the solution derived in Step 5 should include the cost of migration of tasks. If the predicted schedule length produced by the new schedule is greater than that obtained by the current schedule, continue monitoring the grid resources (Step 3). The cost of migration of a task involves the time needed to complete the execution, as well as the time to transfer data. A task is only worth moving if a reduction in execution time compensates for the cost;
- *Step 8* Migrate tasks to the designated hosts on the basis of the most recent schedule.

Fig. 3 shows a diagram portraying the procedure for self-adjustment of resource allocation.

The mapping of tasks to grid nodes and their scheduling (Step 1) demand efficient schedulers. Section 3 will introduce eight novel schedulers for dealing with heterogeneous resources in a grid [15]. These schedulers differ in relation to computational complexity and precision of solution but, depending the time interval involved, either one can be used to obtain the best possible solution.

Note that our proposal is not restricted to monoprocessed hosts. Multiprocessor and multicore hosts can be modeled as a set of grid nodes, each representing a single CPU, connected by edges with null cost, so that all the CPUs in a multiprocessor can be considered for scheduling. Fig. 4 illustrates a network with three hosts, one with two, one with three and one with four CPUs.

In Steps 2 and 8, code and data transfer can be executed using existing protocols, such as FTP and GridFTP [16]. In Step 8, it is assumed that it is possible to resume the execution of an interrupted task by using checkpoints. These checkpoints need to be set by the programmer. The entire execution context of a task can be recorded in a file to be sent together with the task code and data when migrating a task, as is done in the approaches defined in [4,10]. Techniques for monitoring the available bandwidth [17] [18], as well for predicting the network capacity with low computational overhead, are also available [19] [20,21] to Step 3.

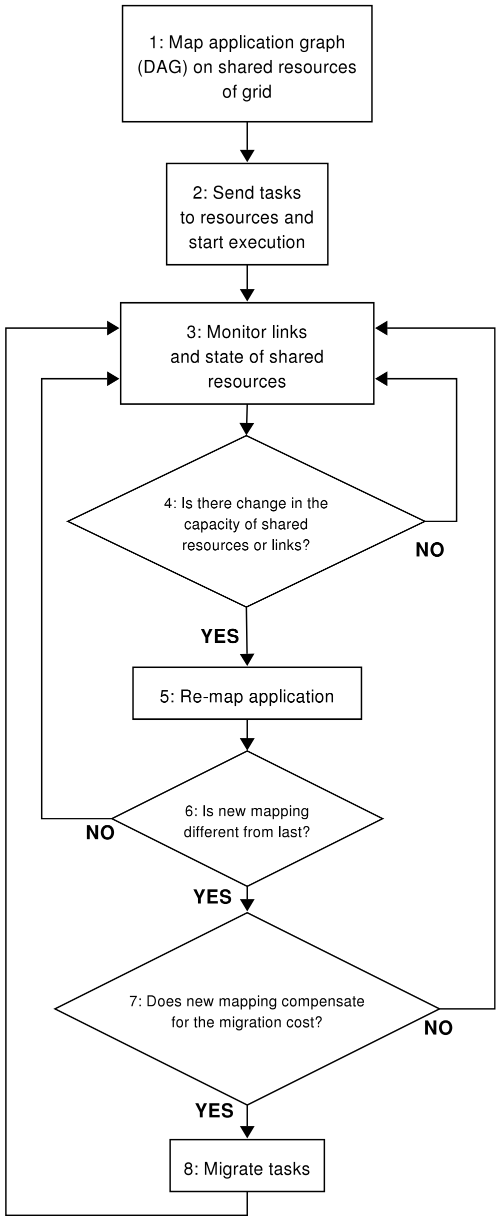
1: Map application graph (DAG) on shared resources of grid
2: Send tasks to resources and start execution
3: Monitor links and state of shared resources
4: Is there change in the capacity of shared resources or links?
NO
YES
5: Re-map application
6: Is new mapping different from last?
NO
YES
7: Does new mapping compensate for the migration cost?
NO
YES
8: Migrate tasks

Fig. 3. A flow diagram of the procedure for grid self-adjustment.

The same schedulers used for the initial scheduling of an Application (Step 1) can be used for the rescheduling and migration of tasks whenever changes in availability of resources are detected (Steps 5, 6 and 7). Rescheduling decisions consider resource availability and current execution status, as well as the initial schedule. Algorithm 1

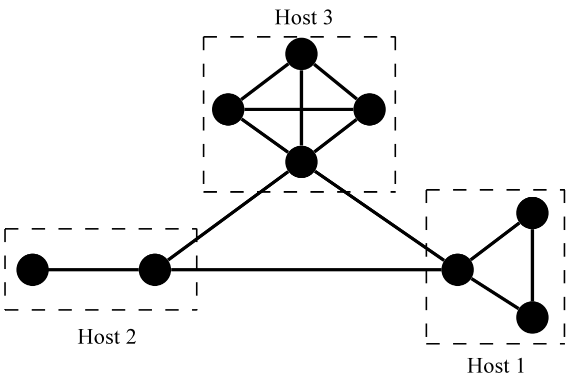

Fig. 4. Graph of a network with multicore hosts.

implements Steps 5–7 and uses the same scheduler used in Step 1.

Algorithm 1 Task rescheduling and migration

```
Input: Previous schedule; DAG with set of tasks $\mathscr{J}$;
   Description of current resource availability status;
   Current time; Scheduler.
1: for each task $i \in \mathscr{J}$ in execution do
2:    Assign the number of instructions already executed
      to the weight of task $i$.
3:    Create a task $i'$ with weight equal to the backlog of
      instructions yet to be executed for $i$.
4:    Move all the outgoing arcs of $i$ to $i'$.
5:    Create an arc $ii'$ with weight equivalent to the
      number of bytes that need to be transferred if task
      $i$ migrates.
6:    Assign to the variable $h$ the id of the host to which
      the task $i$ was mapped prior to the rescheduling
      decision.
7:    Create a new constraint for Scheduler to force task
      $i$ to be scheduled on $h$.
8: end for
9: for each task $k \in \mathscr{J}$ which has either already been
   executed or is presently receiving data from others
   tasks do
10:    Add a constraint to keep the $k$th task at the host to
       which it was initially scheduled.
11: end for
12: Execute the Scheduler with the new constraints and
    the new DAG.
13: for each task $i \in \mathscr{J}$ in execution do
14:    if host to which $i'$ be mapped $\neq$ host to which $i$ was
       mapped prior to rescheduling decision
15:       Migrate task $i$ to the new host.
16:    end if
17: end for
```

Algorithm 1 works on a modified DAG, portraying the evolution of an execution up to a certain time. For each task i in execution, a new task i' representing the current execution status is created. Tasks that have already been executed are kept at the node where they finished. Tasks receiving data from other tasks on which they depend are also kept at the same node. Task i will migrate only if task i' is mapped to a different resource than that to which task i is mapped. The use of this kind of DAG to reschedule the tasks of an application is a notable aspect of our proposal. Such a DAG describes the exact state of processing, thus allowing a more accurate and efficient schedule which will minimize execution time.

However, the proposed procedure do not deal with uncertainties in task demands. Moreover, the programmer must indicate checkpoints for tasks for their rescheduling and migration, as in other approaches [4,10]. This allows the execution of Step 3 in Algorithm 1. If checkpoints cannot be established, the task must be reexecuted when migrated to a different host. In this case, the number of instructions in Step 2 of Algorithm 1 should be zero and the backlog in Step 3 should be the original number of instructions.

The self-adjusting capacity allows great flexibility and can be introduced in middlewares for grids such as [5] [8,6]. Fig. 5 illustrates the introduction of the procedure of self-adjustment into the scheme proposed in [12] represented on both sides of the figure. Note that according to the procedure in [12], once a task is scheduled to a host, the only monitoring involved is related to the execution of the task, which can result in the task migration in the case of performance degradation. The central portion of the figure is the procedure introduced here, and it replaces the dashed part of the scheme in [12].

Other proposals [22] use a single DAG in a attempt to enhance the fairness of resource sharing when several different applications are submitted to a grid. Note that nothing precludes the use of the proposed procedure with a single DAG representing multiple applications.

3. Grid schedulers

The scheduling of tasks to heterogeneous resources is a well-known NP-hard problem, and various sub-optimal solutions which can be reached in a reasonable amount of time have been proposed. This section introduces eight different schedulers for the grid scheduling problem. They differ in the length of the schedule produced, as well as in the time required to derive them. Such diversity allows the selection of the best possible schedule for a given set of time requirements. Fast schedulers can be employed in Step 1, whereas those which give schedules closer to the optimum one can be used in Steps 5–7, since these steps usually involve fewer tasks.

The aim of all the schedulers presented is the minimization of execution time for grid applications under the following restrictions:

- The execution of a task should begin only after the completion of all the other tasks which the task depends on, as well as only after the reception of all data sent by these tasks;
- Each task can be mapped to only one host;
- Two dependent tasks can only be mapped to hosts which have a connecting link (each host is assumed to have a virtual link to itself with zero cost associated with that link);
- Each host can execute only a single task at any time.

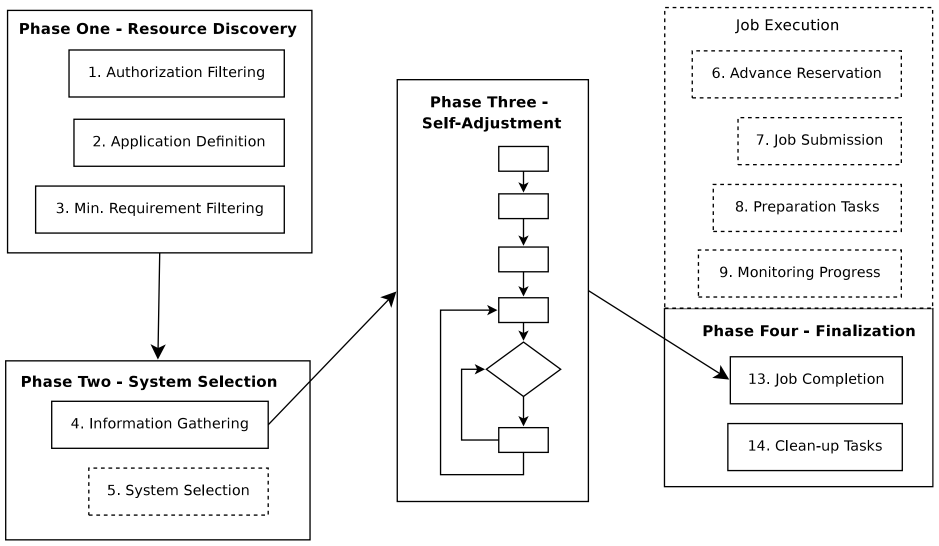
Phase One - Resource Discovery
1. Authorization Filtering
2. Application Definition
3. Min. Requirement Filtering
Phase Two - System Selection
4. Information Gathering
5. System Selection
Phase Three - Self-Adjustment
Job Execution
6. Advance Reservation
7. Job Submission
8. Preparation Tasks
9. Monitoring Progress
Phase Four - Finalization
13. Job Completion
14. Clean-up Tasks

Fig. 5. Inclusion of procedure for self-adjustment in the process shown in Fig. 1.

The schedules produced by six of the eight schedulers proposed are derived from the solution of mixed integer or integer programming problems. Three of these schedulers consider time to be a continuous variable $(\in \mathbb{R}_+)$ whereas the other three consider it as a discrete variable $(\in \mathbb{Z}_+)$. The choice involves a certain trade-off between execution time and the schedule length. Although the discretization of time introduces approximation and a consequent loss of precision, under certain circumstances, this loss may not be significant, and the saving of time can be quite attractive. The exact solution for a integer/mixed integer programming problem for both continuous and discrete time are derived. The other four schedulers are formulated by employing two different relaxation techniques to the exact problems.

The schedulers which consider time as a continuous variable are formulated as a mixed integer programming problem (MIP) whereas those that consider time as a discrete variable are formulated as integer programming problem (IP). In these problems, variables $X_{i,k}$ define the mapping of tasks to hosts; $X_{i,k}$ is 1 if the ith task is mapped to kth host; otherwise, it is 0.

Although solving exact integer and mixed integer programming problems with integrality constraints leads to optimal or quasi-optimal solutions, it may take a very long time. An alternative is the obtainment of partial fractional solutions by considering relaxation of integrality constraints, with the option of conversion of these solutions to integer ones. In this case, the variables $(X_{i,k})$ are defined in the interval $[0,1]$. Techniques for the relaxation of integrality constraints adopt randomized rounding techniques, in which the value of the variable $X_{i,k}$ is the probability of the ith task being mapped to the kth host. Two different randomized rounding techniques were adopted to define two different algorithms. Algorithm 2 solves a linear programming (LP) problem once, with the value of the variables used as probabilities for a series of drawings, each defining a different schedule; the one yielding the shortest schedule is selected as the solution. In Algorithm 3, an iterative randomized rounding procedure is adopted. In each step of this algorithm, an LP is solved, and the task with the highest probability values is definitely mapped to a host. Each one of the iterations of Algorithm 3 ends when no more tasks are left to be mapped to a host. The linear programming solution given as input to both algorithms is the one obtained by relaxation of the integrality constraints.

Algorithm 2. Randomized rounding

```
Input: Relaxation of mixed integer or integer program IP
    to schedule the set of tasks $\mathcal{J}$ in the set of hosts $\mathcal{H}$;
    $P$ = Number of drawings.
Output: Schedule of $\mathcal{J}$ in $\mathcal{H}$.
1: Let $X$ be the solution of the relaxation of IP, where
   $X = (X_{i,k})$.
2: for $P$ times do
3:    for each task $i \in \mathcal{J}$ do
4:       Let the probability of mapping the task $i$ to the
         host $k$ be $X_{i,k}$.
5:       Select a host where the task $i$ should be executed
         based on the previous mapping probability.
6:    end for
7:    Obtain the starting time for each task, considering
      the finishing time of the tasks which it depends on.
8:    Keep this schedule if it is the shortest one.
9: end for
10: Return the shortest schedule.
```

Theorems 1 and 2 establish the time complexity of Algorithms 1 and 2, respectively. Some notations are necessary to understand them. Sets $\mathcal{J}$ and $\mathcal{H}$ are the sets of tasks and hosts, respectively, and $\mathcal{D}$ the set of arcs of the DAG. The time complexity to solve linear programming $\mathcal{P}$ is defined as $\alpha_{\mathcal{P}}$. In this case is considered that it is at least the time complexity to read the problem instance and to set variables $X_{i,k}(\alpha_{\mathcal{P}} = \Omega(|\mathcal{J}| \cdot |\mathcal{H}| + |\mathcal{D}| + |\mathcal{H}|))$.

Theorem 1. *The time complexity of Algorithm 2 is* $O(\alpha_{\mathscr{P}} + P \cdot (|\mathscr{J}| \cdot \log|\mathscr{H}| + |\mathscr{D}| + |\mathscr{H}|))$.

Proof. See Appendix I. □

Theorem 2. *The time complexity of Algorithm 3 is* $O(Q \cdot |\mathscr{J}| \cdot \alpha_{\mathscr{P}})$.

Proof. See Appendix II. □

Algorithm 3. Iterative randomized rounding

```
Input: Relaxation of mixed integer or integer program IP
  to schedule the set of tasks $\mathscr{J}$ in the set of hosts $\mathscr{H}$;
  $Q$ = Number of iterations.
Output: Schedule of $\mathscr{J}$ in $\mathscr{H}$.
1: for $Q$ times do
2:   Let IP be the original mixed integer or integer
     program given in the input.
3:   Let $X$ be the solution of the relaxation of IP, where
     $X = (X_{i,k})$.
4:   for each task $i \in \mathscr{J}$ do
5:     Let the probability of mapping the task $i$ to the
       host $k$ be $X_{i,k}$.
6:     Select a host where the task $i$ should be executed
       based on the previous mapping probability.
7:     Add to the IP the constraint that the task $i$ must be
       mapped to the host $k$.
8:     Let $X$ be a fractional optimum solution of this new
       IP.
9:   end for
10:  Obtain the starting time for each task, considering
     the finishing time of the tasks which it depends on.
11:  Keep this schedule if it is the shortest one.
12: end for
13: Return the shortest schedule.
```

Note that, Algorithm 3 solves a linear programming problem several times. When a linear program is solved after the modification of the boundary of some of the variables, the new linear program is solved much faster (in practice) than was the first version, since the new execution can take advantage of the basis and the information already stored from previous executions of the problem.

The other two schedulers are based on random drawing. The schedule is one of those produced during a series of drawings that minimizes the schedule length. The first step of each iteration of these algorithms is the assignment of an initial value to the variables $X_{i,k}$. The actual starting values constitute the only difference between the two algorithms. In one, it is based on a probability that is uniformly distributed among the hosts, whereas in the other, the probabilities values are set to minimize the execution time of tasks while maximizing resource utilization, and will be denominated "grid aware". In both algorithms, the dependency constraints shown in the DAG, the network topology and the resource capacity are observed. Moreover, these algorithms produce different schedule lengths itself as well as for their own execution time. The one using "grid aware" initial values tends to run for longer periods, but produces shorter schedule length.

Hosts are labelled from 1 to m, while tasks are identified by labels from 1 to n. Tasks are processed according to a topological order of the input DAG, each with a single input task and a single output one. DAGs failing to satisfy this condition because they have more than one input or output task can be easily modified by considering two null tasks with zero processing time and communication weight [23]. Some characteristics of the DAGs are:

- n: number of tasks ($n \in \mathbb{N}$);
- I_i: processing demand of the ith task, expressed as number of instructions to be processed by the task $i (I_i \in \mathbb{R}_+)$;
- $B_{i,j}$: number of bytes transmitted between the ith task and the jth task ($B_{i,j} \in \mathbb{R}_+$);
- $\mathscr{D}$: set of arcs $\{ij : i < j$ and there exists an arc from vertex i to vertex j in the DAG$\}$;
- s_0: starting time of the input task. For all examples in this paper, $s_0 = 0$.

Moreover, grid resources composed of hosts and links have the following characteristics:

- m: number of existing hosts ($m \in \mathbb{N}$);
- TI_k: time the kth host takes to execute 1 instruction ($TI_k \in \mathbb{R}_+$);
- $TB_{k,l}$: time for transmitting 1 bit on the link connecting the kth host and the lth host ($TB_{k,l} \in \mathbb{R}_+$);
- $\mathscr{N}$: set $\{kl$: host k is linked to host $l\}$. In particular, $kk \in \mathscr{N}$ for any host k and if $kl \in \mathscr{N}$ then we also have $lk \in \mathscr{N}$;
- $\delta(k)$: set of hosts linked to the kth host in the network, including the host k itself.

Moreover, $T_{\max}$, is the time that the application would take to execute serially all the tasks in the fastest host, i.e., $T_{\max} = \min TI_k \sum_{i=1}^{n} I_i$, where $\min TI_k$ is the lowest value of TI_k for any host k. $\mathscr{J} = \{1, \ldots, n\}$ is the set of existing tasks of an application and $\mathscr{H} = \{1, \ldots, m\}$ is the set of hosts.

The remainder of this section is organized as follows. Section 3.1, introduces a formulation using continuous time variables whereas Section 3.2 presents the formulation with discrete time variables. Section 3.3 introduces a scheduler based on random drawing that assigns uniform probability values to the initial values. Section 3.4 presents the algorithm which assigns values to the initial probabilities that takes the grid constraints into consideration. Section 3.5 provides an evaluation of the schedulers.

3.1. MIP Formulation with time as a continuous variable

This approach adopts a mixed integer programming formulation for the grid scheduling problem. The final scheduling is established by the value of the following variables:

- $X_{i,k}$, which has the value 1 if the ith task is mapped to the kth host; otherwise it is 0 ($X_{i,k} \in \{0, 1\}$);
- s_i, which sets the starting time of the ith task ($s_i \in \mathbb{R}_+$).

The problem formulation is given by

$$\text{Minimize} \quad \left(I_n \sum_{k=1}^{m} TI_k X_{n,k}\right) + s_n$$

such that

$$s_i \geqslant s_0 \quad \text{for } i \in \mathscr{J}; \tag{C1}$$

$$s_j \geqslant s_i + \sum_{k\in\mathscr{H}} \left[(I_i TI_k X_{i,k}) + \sum_{l\in\delta(k)} (B_{i,j} TB_{k,l} VA_{i,k,j,l}) \right]$$
$$\text{for } i,j \in \mathscr{J},\ ij \in \mathscr{D}; \tag{C2}$$

$$s_j \geqslant s_i + \sum_{k\in\mathscr{H}} (I_i TI_k VA_{i,k,j,k}) - y(1 - P_{i,j})$$
$$\text{for } i,j \in \mathscr{J},\ i \neq j,\ ij \notin \mathscr{D},\ ji \notin \mathscr{D}; \tag{C3}$$

$$s_i \geqslant s_j + \sum_{k\in\mathscr{H}} (I_j TI_k VA_{j,k,i,k}) - yP_{i,j}$$
$$\text{for } i,j \in \mathscr{J},\ i \neq j,\ ij \notin \mathscr{D},\ ji \notin \mathscr{D}; \tag{C4}$$

$$\sum_{k\in\mathscr{H}} X_{i,k} = 1 \quad \text{for } i \in \mathscr{J}; \tag{C5}$$

$$\sum_{l\in\delta(k)} VA_{i,k,j,l} = X_{i,k}$$
$$\text{for } i,j \in \mathscr{J},\ ij \in \mathscr{D},\ k \in \mathscr{H}; \tag{C6}$$

$$2VA_{i,k,j,l} \leqslant X_{i,k} + X_{j,l}$$
$$\text{for } i,j \in \mathscr{J},\ ij \in \mathscr{D},\ k,l \in \mathscr{H},\ kl \in \mathscr{N}; \tag{C7}$$

$$VA_{i,k,j,l} - X_{i,k} - X_{j,l} \geqslant -1$$
$$\text{for } i,j \in \mathscr{J},\ ij \in \mathscr{D},\ k,l \in \mathscr{H},\ kl \in \mathscr{N}; \tag{C8}$$

$$2VA_{i,k,j,k} \leqslant X_{i,k} + X_{j,k}$$
$$\text{for } i,j \in \mathscr{J},\ i \neq j,\ ij \notin \mathscr{D},\ ji \notin \mathscr{D},\ k \in \mathscr{H}; \tag{C9}$$

$$VA_{i,k,j,k} - X_{i,k} - X_{j,k} \geqslant -1$$
$$\text{for } i,j \in \mathscr{J},\ i \neq j,\ ij \notin \mathscr{D},\ ji \notin \mathscr{D},\ k \in \mathscr{H}; \tag{C10}$$

$$VA_{i,k,j,l}, X_{i,k}, P_{i,j} \in \{0,1\} \quad \text{for } i,j \in \mathscr{J},\ k,l \in \mathscr{H}. \tag{C11}$$

The relaxation of the above problem consists of replacing $\{0,1\}$ in the constraints (C11) by the interval $[0,1]$.

The constraints in (C1) state that all tasks must start after time s_0. The constraints in (C2) specify that a task will start only after all tasks dependent on it have been completed and the relevant data transferred. Constraints (C3) and (C4) state that if two independent tasks are scheduled to the same host, one of them will be fully executed before the start of the other. The binary variable $P_{i,j}$ has value 1 if the ith task is executed first and 0 if the jth task is executed first. The constant y is a large positive number (e.g., $T_{\max}$). Constraint (C5) states that the tasks must be scheduled to some host (k). Constraint (C6) specifies that there should be a single tuple (i,k,j,l) such that the ith and jth tasks are scheduled to the kth and to the lth hosts, respectively.

Constraints (C7)–(C9) and (C10) determine that $VA_{i,k,j,l}$ is 1 if and only if $X_{i,k} + X_{j,l}$ is 2. The value of these two variables indicates that tasks with a dependency relationship should be mapped to interconnected hosts.

This formulation involves $O(m^2n^2)$ constraints, and $O(m^2n^2)$ variables. The scheduler based on the exact solution of this problem involving mixed integer programming with a continuous time variable is denominated MIPCT. There are two schedulers based on the relaxation of MIPCT, one involving Algorithm 2 based on randomized rounding (CT-RR) and the other using Algorithm 3 based on iterative randomized rounding (CT-IRR).

Since MIPCT does not make any approximation, its execution time is quite larger than the execution time of the others schedulers. Although this make MIPCT inappropriate to real applications, the schedule it produces is quite useful for comparing with the schedule produced by the other schedulers.

Mixed integer programming problems and integer programming problems are solved by using linear programming formulations. There are many fast algorithms and methods for solving LP problems; one of the most used is the simplex method [24]. Although this method does not lead to polynomial time complexity algorithms (it is exponential in the worst case), it is very fast in practice. Worst- and average-case analyse of algorithms to solve LP problems lead to time complexity bounds that are still high compared to the real behaviour of these algorithms. The number of pivots required by the simplex methods is generally linear, or at most polynomial. Experimental work has shown that, in general, the number of pivot steps is bounded by $3v$ [25–27], with v being the number of variables in the LP. Karmarkar [28] presented a polynomial time algorithm using interior point methods. This method has obtained faster solutions than the simplex method when resolving various LP problems [29,30]. Indeed, benchmarks for LP solvers can be obtained at http://plato.asu.edu/bench.html, where it can be verified that even large LP problems with thousands of variables and constraints can be solved in seconds/minutes.

3.2. IP formulation with time as a discrete variable

This formulation considers discrete intervals of time and treats the scheduling problem as an integer programming problem. For convenience, the following notation is used: $\mathscr{T} = \{1, \ldots, T_{\max}\}$. The schedule is established by the value of the following variables:

- $x_{i,t,k}$: Binary variable that assumes a value of 1 if the ith task finished at time t in the host k; otherwise this variable assumes a value of 0;

The integer programming problem is formulated as follows:

$$\text{Minimize} \quad \sum_{t\in\mathscr{T}} \sum_{k\in\mathscr{H}} t x_{n,t,k}$$

$$\text{such that} \quad \sum_{t\in\mathscr{T}} \sum_{k\in\mathscr{H}} x_{j,t,k} = 1 \quad \text{for } j \in \mathscr{J}; \tag{D1}$$

$$x_{j,t,k} = 0 \quad \text{for } j \in \mathscr{J},\quad k \in \mathscr{H},$$
$$t \in \{1, \ldots, \lceil I_j TI_k \rceil\}; \tag{D2}$$

$$\sum_{k\in\delta(l)} \sum_{s=1}^{\lceil t - I_j TI_l - B_{i,j} TB_{k,l} \rceil} x_{i,s,k} \geqslant \sum_{s=1}^{t} x_{j,s,l}$$
$$\text{for } i,j \in \mathscr{J},\ ij \in \mathscr{D},\quad \text{for } l \in \mathscr{H},\ t \in \mathscr{T}; \tag{D3}$$

$$\sum_{j \in \mathscr{J}} \sum_{s=t}^{\lceil t+I_j TI_k - 1 \rceil} x_{j,s,k} \leqslant 1$$
$$\text{for } k \in \mathscr{H}, \quad t \in \mathscr{T}, \quad t \leqslant \lceil T_{\max} - I_j TI_k \rceil; \tag{D4}$$
$$x_{j,t,l} \in \{0,1\} \quad \text{for } j \in \mathscr{J}, \quad l \in \mathscr{H}, \ t \in \mathscr{T}. \tag{D5}$$

The relaxation of the discrete time formulation consists of changing the set {0,1} of the constraints in (D5) to the interval [0,1].

The constraints in (D1) specify that a task must be executed at one time in a single host. The constraints in (D2) determine that a task (j) cannot terminate until it has been executed in the host k. The constraints in (D3) establish that if the ith task executes in the lth host before the jth task does, and that the jth task is finished at time t, then the time when the ith task finished its execution is at most t minus the execution time of the jth task minus the time needed to transfer data between these two tasks. The constraints in (D4) establish that there is at most one task in execution at any one host at a specific time.

The accuracy of the results obtained by using this formulation depends on the interval width used in the discretization of the timeline. The wider the interval is, the faster the execution; but, the lower the accuracy.

This formulation involves $O(n^2 m T_{\max})$ constraints and $nmT_{\max}$ variables. The scheduler based on an exact solution of the integer programming with a discrete time variable is denominated as IPDT. Again, two versions of schedulers with relaxation are presented, one involving Algorithm 2 with randomized rounding (DT-RR) and the other using Algorithm 3 with iterative randomized rounding (DT-IRR).

3.3. Random drawing with uniform probabilities

The seventh scheduler is based on an algorithm involving random probabilities of task assignment to hosts. It uses an uniform probability distribution to assign tasks to hosts. The distribution is subject to dependency relationships established in the tasks DAG, the network topology and resources capacity. The scheduler is denoted as RDU and the algorithm is shown in Algorithm 4.

Theorem 3 gives the time complexity of Algorithm 4.

Algorithm 4. Random drawing with uniform probability distribution

```
Input: DAG with set of tasks $\mathscr{J}$; Description of current
    resource availability status $\mathscr{H}$; $P$ = Number of
    drawings.
Output: Schedule of $\mathscr{J}$ in $\mathscr{H}$
1: for $P$ times do
2:    Set the probability value for scheduling each task to
      a host as $1/m$.
3:    for each task $i \in \mathscr{J}$ do
4:       Assign randomly a host $k \in \mathscr{H}$ to the task $i$, using
         the previously defined probability value.
5:       Normalize the probability values of the tasks
         dependent on the $i$th task, considering that this
         probability for a tasks dependent on the $i$th task is
         null if it is assigned to a host with no link to the
         host to which the $i$th task is mapped.
6:       Compute the starting time of the $i$th task
         considering the finishing time of all tasks
         dependent on it, as well as the time required to
         transfer data from the dependent task to the $i$th
         task.
7:    end for
8:    Keep this schedule in case it is the shortest one
      produced so far.
9: end for
10: Return the shortest schedule.
```

Theorem 3. *The time complexity of Algorithm 4 is* $O(P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$.

Proof. See Appendix III. □

3.4. Drawing using distribution involving grid-aware probability values

This scheduler differs from the one in the previous subsection by the probability values used for the assignment of tasks to hosts. The following rules are considered to derive the probability values:

1. The probability that a task will be executed in a given host is proportional to the processing rate of all available hosts;
2. The probability of execution of a task by a given host is proportional to the number of links connecting it to other hosts, as well as to their available bandwidth;
3. The larger the number of edges in a task, the higher is the probability that the task will be assigned to a host with large number of links connecting it to other hosts;
4. The greater the amount of data a task needs to transfer, the higher is the probability that the task will be assigned to a host with high capacity links;
5. The larger the number of instructions involved in a task, the higher is the probability that the task will be assigned to a host with a large available processing rate;
6. The lower the level of a task in a DAG, the higher is the probability that the task will be assigned to a host with a high available processing rate, a large number of links and high capacity links (Moreover, the earlier the termination of tasks at the lower levels of the DAG, the earlier the other tasks can finish and, consequently, the shorter the makespan of the application).

The set of rules given above is denominated "set of rules 1" in Algorithm 5. The first two rules define the initial probability of mapping the ith task to the kth host, given by:

$$X_{i,k} = \left(\frac{\frac{1}{TI_k}}{\sum_{j=0}^{m} \frac{1}{TI_j}} \times \frac{1}{3} \right) + \left(\frac{|\delta(k)| - 1}{\sum_{j=1}^{m} |\delta(j)| - m} \times \frac{1}{3} \right) + \left(\frac{\sum_{l \in \delta(k) - \{k\}} \frac{1}{TB_{k,l}}}{\sum_{j=1}^{m} \sum_{l \in \delta(j) - \{j\}} \frac{1}{TB_{j,l}}} \times \frac{1}{3} \right). \tag{1}$$

Underlying these two rules is the idea that the execution time will be shorter if tasks are allocated to the hosts with the highest available processing capacity and bandwidth. However, if the criteria used were to be limited to grid resources, hosts with greater availability of processing rates and bandwidths would be utilized all the time, whereas those with less capacity would remain idle. To avoid such an unbalance, which would lead to unsatisfactory results, the characteristics of tasks also need to be considered, as in list scheduling approaches [23,31]. Consequently, the probability value in Eq. (1) is redefined for each task considering the last four rules defined above.

The DG scheduler is presented in Algorithm 5.

Theorem 4 gives the time complexity of Algorithm 5.

Theorem 4. *The time complexity of Algorithm 5 is* $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$.

Proof. See Appendix IV. □

Algorithm 5. Drawing using distribution involving "grid-aware" probability values

```
Input: DAG with set of tasks $\mathscr{J}$; Description of current
  resource availability status $\mathscr{H}$; $P$ = Number of
  drawings.
Output: Schedule of $\mathscr{J}$ in $\mathscr{H}$
1: for $P$ times do
2:   Set the probability for scheduling a task to a host on
     the basis of the "set of rules 1".
3:   for each task $i \in \mathscr{J}$ do
4:     Select randomly the host $k \in \mathscr{H}$ for the execution
       of the $i$th task.
5:     Normalize the probability values of the tasks
       dependent on the $i$th task.
6:     Compute the starting time of the $i$th task
       considering the finishing time of all tasks
       dependent on it, as well as the time required to
       transfer data from these tasks to the $i$th task.
7:   end for
8:   Consider this schedule if it produces the shortest
     execution time so far.
9: end for
10: Select the shortest schedule.
```

3.5. Comparison of scheduler efficiency

Various network topologies and task DAGs were used to compare the schedulers proposed here. Results of the experiments involving the DAG shown in Fig. 6 and the DAG shown in Fig. 7 are representative of those obtained for other experiments and will be presented in this section. The first DAG represents the `Griz` application, a remote rendering application [13] and the second represents the `Montage` application, an application for the processing of astronomy images [32]. These two DAGs will be referred as `Griz` and `Montage` DAGs, respectively.

The criteria used for comparison are the speedup (the ratio between the time to a serial execution of the tasks in the processor with the greatest available processing rate and the time for task execution using a specific schedule) and the execution time required to produce that schedule. A workstation equipped with a Pentium 4, 3.2 GHz CPU with 2 GB RAM was used in the experiments. The software `Xpress` was employed to solve the integer and mixed integer programming problems. Computer programs were developed using the `C` language.

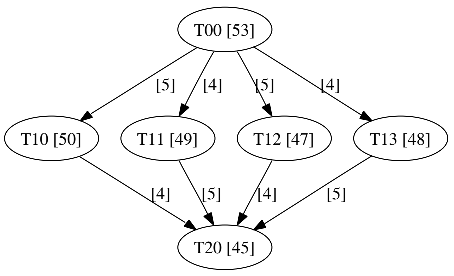

Fig. 6. DAG of tasks of the `Griz` application.

Various topologies were generated using the Doar–Leslie method [33] by changing the number of hosts, the network connectivity (vertex degree) and the ratio between the number of network links (edges) with low bandwidth availability and with high bandwidth availability. This method generates graphs which are similar to real network topologies. It requires as input the number of nodes ($\in \mathbb{N}^*$), the ratio between the number of longest and shortest edges ($\in (0, 1]$) and the connectivity of the graph nodes ($\in (0, 1]$). The length of the edges is related to the weights of the edges. In this paper, the weight of the edges refers to bandwidth availability. Values of connectivity close to 1 gives complete graphs.

If not stated otherwise, in the experiments using the `Griz` DAG, the network used has 50 hosts, network degree 0.5 and ratio between longest and shortest edge 0.9. The processing rate of the hosts follows a uniform probability distribution function in the interval (0.4,2]. The capacity of the network links varied in the interval (0,5], according to the Doar–Leslie method. In the experiments using the `Montage` DAG, if not stated otherwise, the number of hosts is 25. The node degree of the network, the ratio between longest and shortest edge, the processing rate of hosts and the capacity of the network links are the same as in the experiments using the `Griz` DAG. The weights of the DAG arcs in Figs. 6 and 7 were in the interval [4,5], whereas the weight of the vertices varied in the interval [45,54]. Furthermore, except for Algorithm 3, the number of random selections (P) is 10,000. For this algorithm, the number of random selections (Q) is 1, since long execution times were experienced with other values.

For schedulers which consider time as a discrete variable, it is advisable to use a discretizing value corresponding to a fraction of the serial execution time of the DAG. Preliminary experiments suggest that 6.25%, corresponding to a time interval of 8 min for the experiments using the `Griz` DAG and of 16 min for the experiments using the `Montage` DAG, would be an appropriate value.

3.5.1. Results of experiments involving the `Griz` DAG

Tables 1 and 2 show the results of varying number of nodes (hosts). Table 1 presents the performance of the pro-

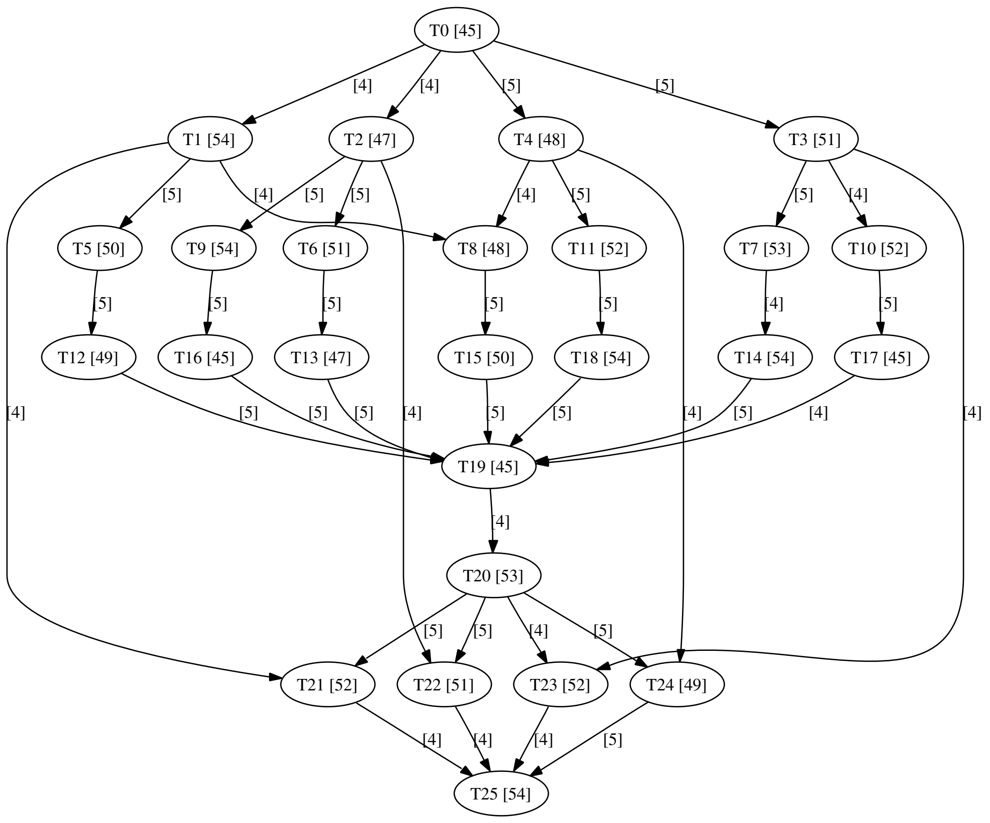

Fig. 7. DAG of tasks of the `Montage` application.

Table 1
Speedup as a function of number of hosts

2 * Hosts	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
10	92.62%	77.71%	99.07%	79.17%	97.26%	78.18%	**1.289432**
40	87.20%	80.04%	**1.251140**	84.59%	97.53%	83.12%	93.29%
70	89.65%	69.34%	92.32%	69.84%	**1.443852**	86.04%	99.28%
100	78.66%	65.34%	94.72%	72.87%	**1.530383**	72.80%	89.62%
130	83.31%	69.40%	96.12%	70.55%	**1.440966**	75.39%	87.31%
160	62.23%	62.11%	91.33%	68.32%	**1.610028**	74.43%	73.89%
190	62.52%	62.30%	81.87%	66.20%	**1.605029**	70.09%	78.31%

Table 2
Execution time as a function of number of hosts

2 * Hosts	Execution time (s)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
10	0.12	0.09	0.30	0.09	0.12	0.09	**0.08**
40	0.74	0.81	**1.13**	1.02	3.04	0.62	0.60
70	3.21	3.79	2.70	1.83	**1.83**	1.82	1.78
100	9.91	10.98	4.38	3.04	**2.88**	3.47	3.33
130	18.43	20.16	7.71	5.46	**4.64**	5.92	5.81
160	40.31	43.18	11.37	12.37	**7.54**	8.69	8.48
190	72.29	76.60	13.24	14.25	**8.04**	11.97	11.62

posed schedulers as a function of the number of hosts. The performance of MIPCT is not shown since, it requires much longer execution time when compared to the other schedulers, as expected. For a 40 host network, for example, MIPCT took over one hour to generate a schedule, whereas IPDT took 3.04 s. The schedule producing the largest speedup for each number of hosts is written in bold. The ratio between other speedup values and the largest one (100%*(speedup/largest_speedup)) is shown as percentage in the table. IPDT produced the largest speedup for most of the experiments, followed by DT-RR. Schedulers based on the relaxation in Algorithm 3 (CT-IRR and DT-IRR) produced the smallest speedup among the schedulers based on both integer and mixed integer programming. This poor performance can be explained by the single random selection of the mapping probabilities in Algorithm 3 ($Q = 1$). For schedulers based on random drawing, DG provides better schedules than does RDU in six out of the seven studied cases, since the initial probability values of the former consider both grid and task constraints. In most of the cases, the CT-RR produced worse results than those produced by the DG scheduler, although they were better than those given by the RDU scheduler.

The execution time of the schedulers, portrayed in Table 2, increases as the number of hosts increases. The execution time of schedulers using a mixed integer formulation increases much more rapidly than does that of integer programming. For example, for a grid with 10 hosts, the execution times of CT-IRR and DT-IRR are about 0.9 s. For a 190-host network, the execution time of CT-IRR is 76.60 s while the execution time of DT-IRR is only 13.24 s. This illustrates the impact that the discretization of time has on decreasing the execution time of LPs. In contrast to what was expected, the execution time of the IPDT scheduler is not always longer than that of the schedulers based on relaxation or drawings. It is a mere consequence of the simplicity of the `Griz` DAG.

Table 3 shows the speedup and Table 4 shows the execution time of the proposed schedulers as a function of network connectivity. This connectivity is expressed as a number in the interval [0,1], a fully connected network having connectivity of 1.0. As in the experiments reported in Tables 1 and 2, IPDT, DG and DT-RR produce the best schedules. DG did generated four of the largest speedups, IPDT did generated two of the largest speedups and DT-RR generated one of the largest speedups. Again, the schedulers based on Algorithm 3 (CT-IRR and DT-IRR) provided the smallest speedup. When the connectivity increases, the execution time typically decreases more than it does when the number of hosts increases.

Tables 5 and 6 shows the performance of the schedulers as a function of the ratio between the number of the longest and the number of the shortest edges. The best schedules were produced by the IPDT, DT-RR and DG schedulers. The use of IPDT led to the largest speedups. Except for the schedules produced by IPDT, the longer the schedule, the longer was the execution time, although there is no clear pattern involving an increase in execution time as a function of the ratio between the number of longest and shortest edges.

Table 4
Execution time as a function of network connectivity

2 * Conect.	Execution time (s)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
0.10	0.74	0.63	1.47	0.63	6.30	1.58	**1.58**
0.22	0.85	0.77	1.39	0.83	6.95	1.48	**1.45**
0.34	0.93	0.96	1.65	0.95	1.10	1.32	**1.30**
0.46	1.17	1.25	2.10	1.54	1.46	1.11	**1.04**
0.58	1.15	1.28	1.92	2.20	**1.35**	1.00	0.93
0.70	1.43	1.78	2.15	2.75	**1.71**	0.42	0.40
0.82	1.56	2.03	**1.74**	1.50	2.29	0.12	0.11

From the results found in those experiments, the scheduler which generated the largest speedup was the IPDT, but the execution time was not as high as expected, given the simplicity of the `Griz` DAG. Schedulers which consider time as a real variable and apply the relaxation algorithms (CT-RR and CT-IRR) did not produce good results. The DT-RR scheduler, which uses Algorithm 2, produced results similar to those of the IPDT scheduler. For various cases, the DG scheduler produced schedules similar to those of the IPDT, but execution times were slightly longer.

3.5.2. *Results of experiments involving the `Montage` DAG*

Tables 7 and 8 show the scheduler performance as a function of the number of hosts of the grid. Results of the schedulers CT-RR, CT-IRR, IPDT and MIPCT are not shown, given the long time of execution with the `Montage` DAG.

As observed in the experiments involving the `Griz` DAG, the DT-RR and the DG schedulers produced the best speedup values and the DT-IRR scheduler the worst one, as can be seen in Table 7. Moreover, the RDU scheduler produces schedules inferior to those provided by the DG scheduler.

The execution time of the schedulers grows with an increase in the number of hosts, as expected. In contrast to the results obtained in the experiments involving the `Griz` DAG, the execution time of schedulers based on linear programming was considerably longer than of those based on drawings. The best schedules were produced by the DT-RR in six out of the seven cases. However, this scheduler took the second longest time to produce the desired schedule. Overall, the DG scheduler represents a good trade-off between the quality of the schedule and the execution time for large DAGs such as the one displayed in Fig. 7.

Table 3
Speedup as a function of network connectivity

2 * Conect.	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
0.10	72.07%	72.07%	85.94%	84.90%	99.84%	95.85%	**1.388938**
0.22	72.97%	72.97%	96.23%	76.37%	94.61%	87.15%	**1.378274**
0.34	70.69%	70.69%	98.79%	73.51%	93.37%	91.51%	**1.423590**
0.46	68.69%	68.69%	89.14%	99.78%	99.50%	89.11%	**1.459943**
0.58	69.85%	69.85%	98.96%	69.99%	**1.440589**	87.73%	96.79%
0.70	64.51%	64.51%	97.60%	65.51%	**1.550246**	83.41%	87.41%
0.82	90.24%	75.41%	**1.335969**	85.80%	74.85%	80.97%	90.85%

Table 5
Speedup as a function of the ratio between the number of longest and shortest edges

2 * Ratio	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
0.20	69.10%	69.10%	**1.458326**	73.50%	95.25%	80.33%	83.36%
0.30	80.02%	80.02%	**1.256459**	83.85%	79.59%	91.36%	93.10%
0.40	63.02%	63.02%	85.87%	65.92%	**1.592708**	81.53%	87.70%
0.50	74.29%	74.29%	91.52%	75.57%	**1.356276**	79.16%	85.01%
0.60	70.22%	70.22%	81.55%	87.15%	**1.431088**	70.22%	82.27%
0.70	65.12%	65.12%	87.12%	87.12%	**1.540268**	74.99%	81.60%
0.80	68.75%	68.75%	92.11%	82.70%	**1.455920**	80.78%	84.55%

Table 6
Execution time as a function of the ratio between the number of longest and shortest edges

2 * Ratio	Execution time (s)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	IPDT	RDU	DG
0.20	1.09	1.20	**1.51**	0.83	0.72	1.09	0.97
0.30	1.07	1.12	**1.43**	0.90	0.80	1.12	1.06
0.40	1.13	1.23	1.80	1.67	**1.17**	1.04	0.97
0.50	1.08	1.16	1.51	0.66	**0.69**	1.13	1.11
0.60	1.18	1.30	1.43	1.33	**0.66**	0.96	0.93
0.70	1.15	1.29	1.53	0.93	**0.71**	1.04	1.01
0.80	1.14	1.27	1.55	1.17	**0.83**	1.02	0.98

Table 7
Speedup as a function of number of hosts

2 * Hosts	Speedup			
	DT-RR	DT-IRR	RDU	DG
10	96.95%	69.87%	88.59%	**1.470900**
15	**2.110896**	73.53%	91.92%	93.96%
20	**1.556400**	66.08%	74.68%	84.09%
25	**1.631350**	81.40%	82.89%	94.05%
30	**1.617550**	62.04%	84.34%	95.73%
35	**1.375990**	94.15%	82.26%	81.31%
40	**1.577601**	66.77%	77.33%	80.17%

Table 8
Execution time as a function of number of hosts

2 * Hosts	Execution time (s)			
	DT-RR	DT-IRR	RDU	DG
10	7.94	99.16	0.31	**0.33**
15	**31.36**	909.42	0.61	0.60
20	**19.72**	445.98	0.88	0.78
25	**39.75**	287.63	1.35	1.29
30	**49.00**	13004.42	1.82	1.87
35	**42.61**	2746.88	2.36	2.29
40	**69.82**	12929.98	2.93	2.88

Tables 9 and 10 show, respectively, the speedup and execution time as a function of network connectivity. An outstanding performance of the DT-RR and DG schedulers was also observed in these experiments, although no clear pattern can be identified for the execution time of schedulers based on linear programming. The execution time of schedulers based on drawing, however, decreases as the network connectivity increases.

The same pattern of performance is observed when the ratio between the number of longest edges and that of shortest edges is varied, as can be seen in Tables 11 and 12. DT-RR provides the best performance, followed by that of DG and DT-IRR, whereas RDU furnishes the worst. Except for DT-IRR, the time of execution did not vary much.

Overall, DG presented the best performance for large DAGs, although the DT-RR scheduler would be a good choice when no strict deadline is imposed for the production of a schedule.

Table 9
Speedup as a function of network connectivity

2 * Connect.	Speedup			
	DT-RR	DT-IRR	RDU	DG
0.10	**1.443350**	69.52%	79.96%	81.09%
0.22	**1.354714**	75.79%	94.36%	95.42%
0.34	**1.397436**	72.61%	81.84%	91.25%
0.46	**1.524434**	72.79%	85.50%	85.68%
0.58	**1.626312**	61.90%	76.43%	91.02%
0.70	**1.517555**	72.72%	76.82%	93.64%
0.82	**1.779391**	57.13%	65.51%	83.96%

Table 10
Execution time as a function of network connectivity

2 * Connect.	Execution time (s)			
	DT-RR	DT-IRR	RDU	DG
0.10	**19.87**	80.19	1.82	1.75
0.22	**24.30**	252.68	1.51	1.49
0.34	**31.00**	486.79	1.34	1.36
0.46	**35.42**	1984.50	1.32	1.32
0.58	**30.54**	988.54	1.33	1.31
0.70	**27.18**	407.15	0.67	0.60
0.82	**36.22**	468.96	0.82	0.66

Table 11
Speedup as a function of the ratio between the number of longest and the number of shortest edge

2 * Ratio	Speedup			
	DT-RR	DT-IRR	RDU	DG
0.20	**2.030034**	69.67%	92.87%	96.63%
0.30	**1.509987**	78.44%	81.41%	91.69%
0.40	**1.600044**	68.88%	89.70%	95.34%
0.50	**1.621040**	80.57%	82.21%	87.09%
0.60	99.53%	69.83%	95.06%	**1.480001**
0.70	**1.532955**	68.01%	80.42%	80.60%
0.80	**1.645130**	61.65%	77.98%	83.16%

Table 12
Execution time as a function of the ratio between the number of longest and the number of shortest edge

2 ∗ Ratio	Execution time (s)			
	DT-RR	DT-IRR	RDU	DG
0.20	**48.27**	1783.99	1.38	1.31
0.30	**35.48**	831.14	1.34	1.28
0.40	**29.83**	501.03	1.30	1.14
0.50	**39.13**	2032.30	1.36	1.35
0.60	33.26	752.62	1.39	**1.43**
0.70	**27.30**	273.98	1.41	1.36
0.80	**30.65**	832.58	1.36	1.38

4. Examples of the use of self-adjustment procedure

This section illustrates the use of the procedure which reduces the execution time for grid applications (schedule length) when changes occur in the availability of resources after the execution of the application has begun. A simulator, the `GridSim-NS`, developed at the University of Trento, was used in the experiments. The `GridSim-NS` is actually a module incorporated into the widely used `NS-2` simulator. `GridSim-NS` receives an Application DAG as input and allows users to define a schedule to be employed by this DAG. In the following experiments the schedules were produced by the schedulers introduced in this paper.

The application is the one described in Fig. 2, whereas the grid is illustrated in Fig. 8. The left hand side of Fig. 8 shows the network topology while the right shows the grid nodes. The arc weights in the DAG represent the amount of data to be transferred, in GigaBytes, and the vertex weights represent the quantity of instructions on a 10^{12} scale. The network has 34 hosts arranged around a central host (SRC_0) and the grid has 11 nodes ($SRC_{\{0\ldots10\}}$). The available processing rate of the host SRC_0 is 1600 MIPS, whereas that of all the others is 8000 MIPS. The links connecting SRC_0 to the other hosts have a capacity of 100 Mbps, whereas all the others are limited to 33.33 Mbps. The topology used resembles CERN's LHC Computing Grid. Note that the topology is not centralized around SRC_0 but the hosts can communicate with each other without going through a central node. Moreover, the processing capacity of node SRC_0 is lower than that of the other hosts, which implies a parallel execution of the tasks in the other hosts.

The batch means method was used to obtain a confidence intervals of 95% confidence level. The width of the intervals was less than 5% of the mean value. Confidence intervals are omitted for the sake of visual interpretation.

The first experiment was designed to determine the time required for the application execution under ideal conditions so that it could be used as a standard for comparison. In the second experiment, bandwidth was reduced and all the steps of the procedure for self-adjustment were executed. The third experiment included the increase in resource availability, and the final one evaluated the impact of the frequency of monitoring on the performance.

In the first experiment, the application (Fig. 2) was mapped using the MIPCT scheduler, with a resultant mapping of $0 \rightarrow SRC_0$, $1 \rightarrow SRC_2$, $2 \rightarrow SRC_5$, $3 \rightarrow SRC_8$, $4 \rightarrow SRC_4$, $5 \rightarrow SRC_1$, $6 \rightarrow SRC_9$, $7 \rightarrow SRC_{10}$, $8 \rightarrow SRC_0$. Similar mapping

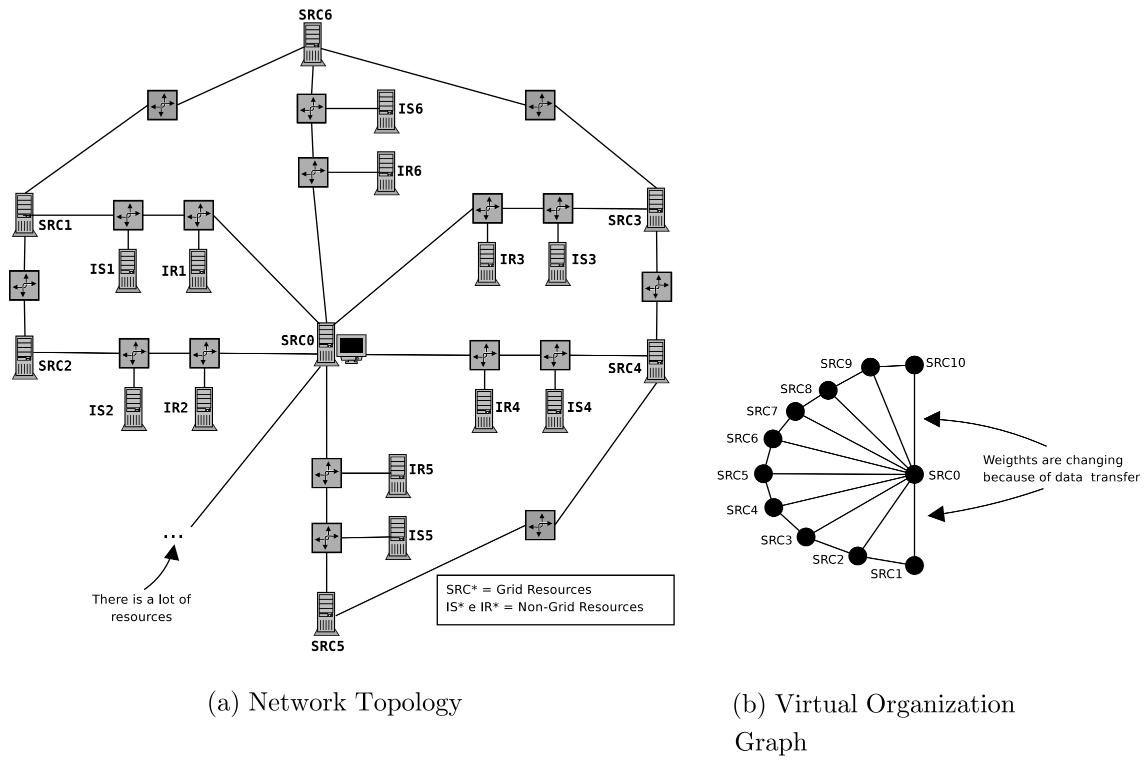

Fig. 8. Grid used in the examples.

could have involved other hosts, since the topology is symmetrical. In the actual schedule derived, Tasks 1–7 start running at the time of 82.66 min, whereas task number 8 starts running at 175.96 min and finishes at 255.96 min.

In the second experiment, the same scenario and initial mapping were used. However, at 90 min, UDP streams with a rate of 90 Mbps were added as interfering traffic between hosts IR_2 and IS_2 and between IR_5 and IS_5. These traffics impact the resource availability between hosts SRC_2 and SRC_0 and between hosts SRC_5 and SRC_0. Monitoring the resources of the grid was carried out every 40 min. (The use of long monitoring intervals reinforce the effectiveness of monitoring the availability of resources). Thus, at the time 120 min, the need to reevaluate the current schedule had become evident. At that time, the DAG for the remaining tasks was modified, by Algorithm 1, to the one shown in Fig. 9. At time 120 min, Algorithm 1 decomposes each task into two other tasks; one of these remains on the host on which it was originally scheduled, with its weight in the new DAG representing the number of instructions already processed. The other is rescheduled, with its weight representing all the instructions remaining to be executed. Moreover, the weight of the edge between these two tasks indicates the quantity of data to be transferred to the host on which the second task will be scheduled.

For this new DAG, the schedule was obtained by the IPDT scheduler. Since the cost involved in task migration includes that of time needed to complete the execution, as well as that required to transfer data, a task is worth moving only if a reduction in time of execution will compensate for this cost. The new schedule determined that Tasks 1 and 2 should be migrated from hosts SRC_2 and SRC_5 to hosts SRC_3 and SRC_6, respectively. These migrations were designed to avoid the interfering traffic for the transfer of 10 GB of data to Task 8. When migrations occur the new execution time was 281 min, which is only 9.34% higher than the one obtained under ideal conditions. If the tasks had not migrated, the execution time would have been 358 min, i.e., an increase of about 27.4%. Figs. 10 and 11 show, respectively, the time of execution of Task 1 and the round trip time (RTT) between SRC_2 and SRC_3 (The usage of CPU and network by Task 2 are similars). These figures illustrate task migration; it can be seen that between the times 120 min and 150 min, no processing activity took place in either of the hosts SRC_2 and SRC_3, since in this interval, migration take place.

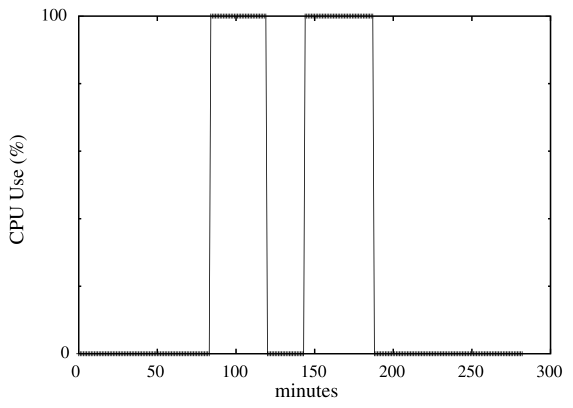

Fig. 10. Use of CPU for Task 1.

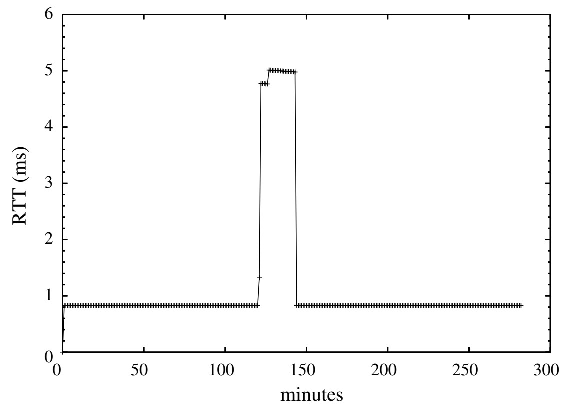

Fig. 11. RTT between SRC_2 and SRC_3.

Where a dynamic scheduling approach to be employed, two options would remain after the completion of tasks 1 and 2: (i) transfer of 10 GB from each task directly to the host SRC_0, leading to an execution time of 358 min, or (ii) transfer of 20 GB to two intermediate nodes (10 GB from

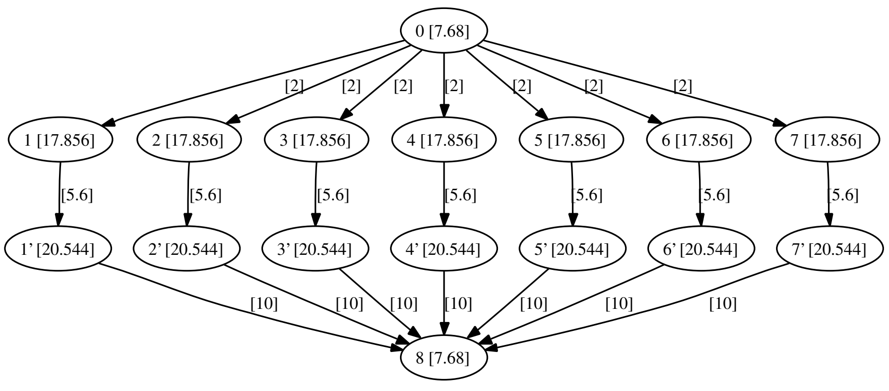

Fig. 9. DAG for migration at 120 min.

SRC_2 to SRC_3 and 10 GB from SRC_5 to SRC_6) before sending to the final destination, SRC_0, thus avoiding congestion of links. This second option would lead to an execution time of ≈300 min. In both cases, the execution time would be longer than that obtained with the present approach.

Moreover, where an adaptive scheduling to be adopted, no migration would be pursued, and consequently the application execution time would be 358 min, which is longer than the time obtained with the present procedure (281 min).

These two examples illustrate the benefit of continuous monitoring of the grid and allowing the triggering of migration at any time to minimize execution time. This is the main difference between the present approach and adaptive and dynamic scheduling.

In the third experiment, resources were added to the grid. Such additions are not necessarily due to the acquisition of new resources, as they may be due to the release of resources by other applications. Fig. 12 illustrates the addition of the host SRC_{16}; the link capacity joining it to host SRC_6 is 1 Gbps, with an available processing rate of 8000 MIPS. Similar hosts were also added to hosts SRC_1 to SRC_{10}. With this extra resource, the execution time decreases to 247 min, which corresponds to a reduction of 3.89% of execution time under ideal conditions. This example shows that task migration should not only be investigated under conditions of a shortage of resources, but also whenever increased resources become available. If, for example, the processing rate available were 4000 MIPS, migration would not be advisable since, execution time would have increased to 291 min if migration were carried out, which is 13.23% higher than that obtained under ideal conditions.

One of the key issues involved in the self-adjustment procedure is the frequency of reevaluation of the adequacy of the schedule under modified resource constraint. To get an idea of the importance of the frequency of this procedure, various simulations were carried out in the fourth experiment. A source of interfering traffic (60 Mbps) was introduced to the same links as in the previous example. Both MIPCT and IPDT schedulers were used for the experiments. First, a simulation with no task migration was run; execution time was 279 min. Then, the recommendations of the scheduler were followed. Table 13 shows the execution time required when task migration is undertaken.

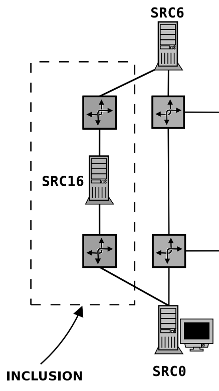

Fig. 12. Inclusion of new resource linked to SRC_6.

Table 13
Execution times as a function of monitoring interval duration (minutes)

Interval	MIPCT	IPDT
100	269 (migration)	269 (migration)
110	275 (migration)	275 (migration)
120	276 (no migration)	281 (migration)
130	276 (no migration)	287 (migration)
140	276 (no migration)	276 (no migration)
150	276 (no migration)	276 (no migration)

It is clear that the frequency of evaluation plays a major role in the execution time. If a long period between changes in resources availability and the decision to migrate a tasks occurs, computation may have progressed to a point in which migration would no longer be an interesting option. Moreover, it can be seen that the IPDT may produce schedules which yield longer execution times than those where no migration is pursued, as can be in the results when intervals of 120 min and 130 min were used. Such imprecision is critical when approaching the "ideal" time for reevaluation due to the approximations introduced by time discretization. In fact, the ideal frequency for reevaluation is system dependent, since it is influenced by the frequency of changes in the resource pool.

5. Related work

Various techniques for monitoring and performance prediction have been employed for systems such as that of the network weather service (NWS) [21], which uses active monitoring techniques, as well as temporal series, to predict performance. One distinct characteristic of the NWS system is its hierarchical monitoring approach. Applications such as those supported by NWS require performance feedback in short periods of time, typically in the order of minutes. Another system for applications which run for long periods is the grid harvest service (GHS) [20] which is more scalable than NWS. In GHS, performance prediction is carried out by neural networks and these predictions are employed to determine task migration. A different monitoring system used in the Wren systems was introduced in [18]; this adopts either active or passive monitoring techniques, depending on the network load. All these proposals for monitoring the status of resources can be incorporated in Steps 3 and 4 of the resources engineering procedure introduced in Section 2. However, the prediction of performance in self-adjustment procedure involves schedulers based on optimization for determining the potential reconfiguration of a grid.

Several self-adjusting systems based on monitoring and task migration have been proposed [5,6,8,7,9,4,11,10] in the literature. Although under different names, all these schemes were designed to minimize the execution time of applications. In all of these, mechanisms have been inserted into existing middlewares and agents for the management of grid applications. The procedure for self-adjustment introduced in this paper differs from all of them, since it uses neither adaptive scheduling nor dynamic scheduling.

Although the proposal in [5] took into consideration the decrease in the performance of an application, no evidence of the effectiveness of the policy for migration was presented. In this approach, an intermediate storage node was used during migration to the final destination. In spite of the flexibility provided by this intermediate node, it can also become a bottleneck. Our proposal does not consider migration to intermediate nodes and consequently does not create such bottleneck.

The GridWay project [6] promotes migration under several circumstances, but neglects the degradation of network performance as a factor for the triggering of task migration; the failure of a link is the only trigger considered. This scheme can generate a large number of task migrations, since it uses a greedy algorithm for fast initial scheduling and then adjusts the schedule in succeeding evaluation steps.

In [8], migration occurs only if the gain in execution time is greater than 30%, although the authors admit that this threshold value may not be the optimal one. The major outstanding difference between the self-adjustment procedure proposed here and that proposed in [8] is that this procedure for self-adjustment computes the migration overhead based on the current grid status, whereas the one proposed in [8] fixes overhead estimation to a constant value.

The AppLeS project [7] uses an adaptive approach for scheduling applications. Besides considering the state of the grid, it also reschedules tasks to improve performance. The present proposal differs from AppLeS because it considers the execution time of the schedulers in the scheduling life time, thus enabling it to work with different deadlines. In [20] it was pointed out that performance degradation can occur when the minimization of the execution time of application is the major goal when using the AppLeS system.

An extension [9] of the GridWay project, which uses Globus middleware, was introduced to make the execution of applications easier and more efficient. Task migration considers the resource availability of hosts as well as the cost of migration in relation to the gain in execution time. However, the approach fails to consider the degradation of network performance as a determining factor for task migration. Moreover, the setting of a threshold value for the gain obtained by task migration limits the potential minimization of the execution time, as reported by the authors.

Other modifications have been proposed for the GridWay system to diminish the time of data transfer by using files shared by tasks residing in the same host [4]. A major disadvantage of this proposal is the fact that it only considers these tasks for scheduling and migration. Such a limitation prevents it from being used for applications with dependent tasks, such as those considered in the present paper. A policy for rescheduling based on the underachievement of the predicted makespan of an application has been proposed in [11]. This policy is robust in relation to imprecisions in the estimate of execution time. As the scheme defined in [12], the policy monitors the progress of the application execution, but it fails to account for changes in resource availability. Thus, the introduction of new resources does not trigger task migration and the consequent improvement in performance. However, the present proposal does take the fluctuation of resource availability into consideration, thus, allowing a dynamic search for the minimum execution time.

In [10], a procedure using rescheduling and task migration to release allocated resources and admit new applications was proposed. It was shown that this proposal presents advantages when compared to those which impose the end of execution tasks as a condition for the admission of new ones. However, the proposal considers the link state only at the time of scheduling of new applications. The authors point out the need for rescheduling as a function of the fluctuation of resource availability, as is carried out by the present proposal.

In summary, the uniqueness of our proposal when compared to existing ones is the consideration of resource availability during all the execution period of an application. Moreover, our proposal is the only one to consider the overhead of task rescheduling in the decision making process.

Various scheduling schemes have been proposed for grids [23,34–37]. The level-branch priority (LBP) [23] algorithm organizes a list of ordered priorities, with the placement of a task depending on its level in the DAG to which it belongs, as well as the number of output edges. This approach is similar to those adopted by the DG scheduler in this paper, but LBP does not consider heterogeneous resources and assumes that all network links to have the same transfer capacity.

The schedule presented in [34] assigns tasks to links rather than to hosts. Moreover, all hosts are assumed to have the same available processing rate which again is not realistic in a grid environment. Various schedulers based on heuristics are presented in [35]. These schedulers produce schedules within a certain time threshold. Results are presented for a single network topology, however, and the effectiveness of the schedulers is compared to a greedy algorithm which does not consider data dependencies in tasks DAGs. The scheduler introduced in [36] was designed to take into account quality of service requirements and consider bandwidth as the only task requirement, ignoring the possibility of data dependency among tasks. Finally, the scheduler proposed in [37] assumes that the time required to transfer data is insignificant in relation to that the spent on processing, making it inappropriate for applications with a large distributed data set shared among tasks.

None of the schedulers proposed in the literature are able to account for heterogeneous grid resources as are the schedulers introduced in this paper. Moreover, none of them works under time constraints. Furthermore, the effectiveness of the schedulers proposed here has been extensively validated in relation to various network topologies and tasks DAGs.

6. Conclusions

Grid networks can accommodate a new generation of users with high computational and data transfer demands.

Although several grid systems already exist, this technology is still in its infancy. One of the major challenges of grids networks is the fluctuation in availability of resources which has a definite impact on the performance of an application. Enabling grid systems for self-adjustment in response to changing scenarios is crucial for autonomy and will facilitate their use.

This paper has introduced a resource allocation approach oriented to applications with dependent tasks so that these applications can adapt their allocation of resources to produce the minimum possible execution time. The proposal differs from others in the literature by considering the network link status in all the phases of the execution of an application (initial scheduling, rescheduling and task migration). The effectiveness of this new procedure has been illustrated by several simulation experiments involving various changes in the simulated grid.

Furthermore, this paper also presented a set of grid schedulers able to deal with heterogeneous grid resources which can be used to produce schedules with different quality given deadline constraints. These schedulers can be executed in parallel to obtain the best possible schedule under specific deadlines. The performance of these schedulers were assessed.

In the future, the dynamic determination of the duration of intervals for reevaluation of schedule needs to be pursued. An interesting topic for future research is the evaluation of the stability of a Grid when several applications employing the self-adjustment procedure are competing for the same resources. Moreover, the resource allocation scheme proposed here shall be introduced into existing systems.

Acknowledgements

The authors would like to sincerely thank the referees for their constructive comments. This work was supported in part by ProNEx–FAPESP/CNPq, CNPq and FAPESP Kyatera.

Appendix I

Theorem 1. *The time complexity of Algorithm 2 is* $O(\alpha_{\mathscr{P}} + P \cdot (|\mathscr{J}| \cdot \log|\mathscr{H}| + |\mathscr{D}| + |\mathscr{H}|))$.

Proof. Algorithm 2 solves the input linear programming $\mathscr{P}$ (relaxation of an MIP or an IP formulation) once in Step 1 in time $O(\alpha_{\mathscr{P}})$. We consider that we can obtain a (pseudo) random number in $[0,1]$ in constant time. Note that the minimum time complexity of Step 1 is at least the time to set each variable of the linear program, $X_{i,k}$, which is at least $O(|\mathscr{J}| \cdot |\mathscr{H}|)$.

After obtaining the values of $X_{i,k}$ for each tasks i and host k, we compute a table $T_{i,k}$ where $T_{i,k} = 0$ if $k = 0$ and $T_{i,k} = T_{i,k-1} + X_{i,k}$ if $k \geqslant 1$. The total time of this step can be executed in time $O(|\mathscr{J}| \cdot |\mathscr{H}|)$ to facilitate the finding of a host (probabilistically) to each task.

Steps 4 and 5 consider a task i and select a host k for this task based on the probabilities given by the linear programming. These steps can be executed in $O(\log|\mathscr{H}|)$ with a binary search in $T_{i,*}$. Therefore, the time completion of these steps is $O(P \cdot |\mathscr{J}| \cdot \log|\mathscr{H}|)$, considering the loops starting in Steps 2 and 3.

One iteration of Steps 7 and 8 can be performed following the topological order of tasks (by DAG precedence) which can be obtained in time $O(|\mathscr{J}| + |\mathscr{D}|)$. To set the starting time of a task i, scheduled on machine k, the algorithm must verify the completion time of the tasks that precede i (at this point they are already scheduled) and the time when machine k becomes available (before the execution of these steps, each machine is set with starting time 0). When we consider all tasks, we have considered all precedences (edges in the DAG) and so, the P executions of Steps 7 and 8 are done in time $O(P \cdot (|\mathscr{J}| + |\mathscr{D}| + |\mathscr{H}|))$. Therefore, the total time complexity of Algorithm 2 is given by $O(\alpha_{\mathscr{P}} + P \cdot (|\mathscr{J}| \cdot \log|\mathscr{H}| + |\mathscr{D}| + |\mathscr{H}|))$. □

Appendix II

Theorem 2. *The time complexity of Algorithm 3 is* $O(Q \cdot |\mathscr{J}| \cdot \alpha_{\mathscr{P}})$.

Proof. The time complexity involved in all executions of Steps 2 and 3 is clearly $O(Q \cdot \alpha_{\mathscr{P}})$.

One iteration of Steps 5–7 can be performed in time $O(|\mathscr{H}|)$ and one iteration of Step 8 can be executed in time $O(\alpha_{\mathscr{P}})$. Therefore, considering all iterations and the fact that $|\mathscr{H}|$ is bounded by $O(\alpha_{\mathscr{P}})$ Steps 5–8 can be executed in time $O(Q \cdot |\mathscr{J}| \cdot \alpha_{\mathscr{P}}))$.

The analysis of Step 10 is similar to the one carried out to Algorithm 2 and it can be performed in time $O(Q \cdot (|\mathscr{J}| + |\mathscr{D}| + |\mathscr{H}|))$.

Since $|\mathscr{D}|$ and $|\mathscr{H}|$ are also bounded by $O(\alpha_{\mathscr{P}})$, the total time complexity of Algorithm 3 is $O(Q \cdot |\mathscr{J}| \cdot \alpha_{\mathscr{P}})$. □

Appendix III

Theorem 3. *The time complexity of Algorithm 4 is* $O(P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$.

Proof. Algorithm 4 iterates P times, and in each iteration it tries to obtain a feasible schedule. One iteration of Step 2 can be performed in $O(|\mathscr{J}| \cdot |\mathscr{H}|)$ time, considering that the probabilities are stored in an appropriate table. So, the total time complexity of this step is $O(P \cdot |\mathscr{J}| \cdot |\mathscr{H}|)$.

One iteration of Step 4 can be executed in $O(|\mathscr{H}|)$ time for each task i. So, this step is performed in $O(P \cdot |\mathscr{J}| \cdot |\mathscr{H}|)$ time, considering all iterations. Step 5 updates the probabilities for each task j for which i must precede. For a task (for which i must precede) the probabilities to connect to a host can be updated in $O(|\mathscr{H}|)$ time. The number of dependencies considered to all tasks in the loop starting in Step 3, is equal to $|\mathscr{D}|$, which is the number of arcs in the DAG. So, the total time complexity for all executions of Step 5 is $O(P \cdot (|\mathscr{D}| + |\mathscr{J}|) \cdot |\mathscr{H}|)$.

In Step 6, it is computed the starting time of the task i. It considers all tasks that task i depends and the finishing time of the host where i is allocated. So, the total time to compute Step 6 for all loops starting in Steps 1 and 3 is $O(P \cdot (|\mathscr{D}| + |\mathscr{J}|))$. Therefore, the total time complexity of Algorithm 4 is given by $O(P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$. □

Appendix IV

Theorem 4. *The time complexity of Algorithm 5 is* $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$.

Proof. The probabilities computed with the first two rules (Section 3.4) define initial probabilities that will be subsequently modified by the application of the last four rules (Section 3.4). To this end, tasks are first ordered by its level in the DAG (task with lower level first) and then rules are applied for each task. Rules are applied from rule 3 to 6 and at last, the values of $X_{i,k}$ are normalized to represent probabilities.

The application of each rule may increase or decrease the value $X_{i,k}$, computed by the previous rules, according to the ranking of host k for some characteristics (e.g. number of connecting links, processing rate, etc.).

For rule 3, the value $X_{i,k}$ is increased, if host k has a large number of connecting links (given by its rank, e.g., host k has rank smaller than $|\mathscr{H}|/2$); otherwise it is decreased. The proportion to be increased/decreased is given by the ratio between the number of arcs incident to task i and the total number of arcs.

For rule 4, the value $X_{i,k}$ is increased, if host k has large capacity links connected to it (given by its rank); otherwise it is decreased. The proportion to be increased/decreased is given by the ratio between the amount of data transferred by i and the total amount of data transferred by tasks.

For rule 5, the value $X_{i,k}$ is increased, if host k has a large processing rate (given by its rank); otherwise it is decreased. The proportion to be increased/decreased is given by the ratio between the number of instructions of i and the total number of instructions for all tasks.

To update the values of $X_{i,*}$ by rule 6, hosts are first ranked by the large values of $X_{i,k}$. Then, the value of $X_{i,k}$ is increased if k is one of the firsts in this ranking (e.g. rank smaller than $|\mathscr{H}|/2$); otherwise it is decreased. The proportion to be increased/decreased is given by $\frac{h-l}{h}$, where h is the higher DAG level and l is the level of task i.

Algorithm 5 differs from Algorithm 4, by the computation of the initial probability values in Step 2. In this case, the probabilities are computed with the "set of rules 1". The first two rules are computed according to Eq. (1). All terms in this equality are independent of i and some summations are independent of k, which leads to a single computation to each distinct value. So, the total time complexity to compute the first two rules is $O(|\mathscr{J}| \cdot |\mathscr{H}| + |\mathscr{H}| + |\mathscr{N}|)$.

Since the ranking computed by rules 3–5 can be performed once for all jobs (by sorting), the time complexity to apply these steps is given by $O(|\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + |\mathscr{D}| + |\mathscr{D}| \cdot |\mathscr{H}|)$. For rule 6, the ranking used must be recomputed for each job i just after the application of the previous rules, since they modify values of $X_{i,k}$; so the time complexity is given by $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + |\mathscr{J}| \cdot |\mathscr{H}|)$.

The time complexity of one execution of Step 2, that computes starting probabilities $X_{i,k}$ for all tasks $i \in \mathscr{J}$ and hosts $k \in \mathscr{H}$, using the "set of rules 1", is given by $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + |\mathscr{D}|)$. If these probabilities are stored, they can be computed once and copied for each iteration of Step 1. So, the time complexity of all executions of Step 2 is $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + |\mathscr{D}| + P \cdot |\mathscr{J}| \cdot |\mathscr{H}|)$.

The remaining steps of Algorithm 5 has the same analysis of Algorithm 4. So, they can be computed in time $O(P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$.

Therefore, Algorithm 5 can be implemented in $O(|\mathscr{J}| \cdot |\mathscr{H}| \log |\mathscr{H}| + |\mathscr{N}| + P \cdot |\mathscr{H}| \cdot (|\mathscr{J}| + |\mathscr{D}|))$ time. □

References

[1] I. Foster, What is the grid? A three point checklist, GRIDToday 1 (6) (2002). <http://www-fp.mcs.anl.gov/foster/Articles/WhatIsTheGrid.pdf> (accessed at 20.10.2006).

[2] H. Casanova, Distributed computing research issues in grid computing, SIGACT News 33 (3) (2002) 50–70.

[3] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.

[4] E. Huedo, R.S. Montero, I.M. Llorente, Experiences on adaptive grid scheduling of parameter sweep applications, in: Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2004, pp. 28–33.

[5] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf, The cactus worm: experiments with dynamic resource discovery and allocation in a grid environment, Int. J. High Performance Comput. Appl. 15 (4) (2001) 345–358.

[6] E. Huedo, R.S. Montero, I.M. Llrorent, An Experimental Framework for Executing Applications in Dynamic Grid Environments, Tech. Rep. 2002-43, NASA Langley Research Center, 2002.

[7] F. Berman, R. Wolski, H. Casanova, W.W. Cirne, H.H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, D. Zagorodnov, Adaptive computing on the grid using AppLeS, IEEE Trans. Parallel Distribut. Syst. 14 (2003) 369–382.

[8] S.S. Vadhiyar, J.J. Dongarra, A performance oriented migration framework for the grid, in: Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGRID'03), 2003, pp. 130–137.

[9] R.S. Montero, E. Huedo, I.M. Llorente, Grid resource selection for opportunistic job migration, in: Proceedings of the 9th International Euro-Par Conference, Springer, Berlin, Heidelberg, 2003, pp. 366–373.

[10] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, J. Pukacki, Dynamic grid scheduling with job migration and rescheduling in the GridLab resource management system, Sci. Program. 12 (2004) 263–273.

[11] R. Sakellariou, H. Zhao, A low-cost rescheduling policy for efficient mapping of workflows on grid systems, Sci. Program. 12 (2004) 253–262.

[12] J.M. Schopf, Ten actions when grid scheduling, in: Grid Resource Management: State of the Art and Future Trends, first ed., Springer, 2003, pp. 15–23.

[13] L. Renambot, T. van der Schaaf, H.E. Bal, D. Germans, H.J.W. Spoelder, Griz: experience with remote visualization over an optical grid, Future Generation Comput. Syst. 19 (6) (2003) 871–882.

[14] D.M. Batista, N.L.S. da Fonseca, F. Granelli, D. Kliazovich, Self-adjusting grid networks, in: Proceedings of the IEEE International Conference on Communications, 2007 – ICC'07, 2007, pp. 344–349.

[15] D.M. Batista, N.L.S. da Fonseca, F.K. Miyazawa, A set of schedulers for grid networks, in: SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing, ACM Press, New York, NY, USA, 2007, pp. 209–213.

[16] M. Cannataro, C. Mastroianni, D. Talia, P. Trunfio, Evaluating and enhancing the use of the GridFTP protocol for efficient data transfer on the grid, in: Proceedings of the 10th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science, vol. 2840, 2003, pp. 619–628.

[17] F. Montesino-Pouzols, Comparative analysis of active bandwidth estimation tools, in: Proceedings of the 5th Passive and Active

Measurement Workshop (PAM 2004), Lecture Notes in Computer Science, vol. 3015, 2004, pp. 175–184.
[18] B.B. Lowekamp, Combining active and passive network measurements to build scalable monitoring systems on the grid, SIGMETRICS Perform. Evaluat. Rev. 30 (4) (2003) 19–26.
[19] J.M. Schopf, L. Yang, Using predicted variance for conservative scheduling on shared resources, in: Grid Resource Management: State of the Art and Future Trends, first ed., Springer, 2003, pp. 215–236.
[20] X. Sun, M. Wu, GHS: a performance system of grid computing, in: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, 2005, p. 228–233.
[21] R. Wolski, N.T. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Generation Comput. Syst. 15 (5–6) (1999) 757–768.
[22] H. Zhao, R. Sakellariou, Scheduling multiple DAGs onto heterogeneous systems, in: Proceedings of the 20h International Parallel and Distributed Processing Symposium (IPDPS 2006), 2006.
[23] D. Ma, W. Zhang, A static task scheduling algorithm in grid computing, in: Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC 2003) – Part II, Lecture Notes in Computer Science, vol. 3033, 2004, pp. 153–156.
[24] G.B. Dantzig, Maximization of linear function of variables subject to linear inequalities, in: T.C. Koopmans (Ed.), Activity Analysis of Production and Allocation, 1951, pp. 339–347.
[25] M.J. Todd, The many facets of linear programming, Math. Program. 91 (3) (2004) 417–436.
[26] R.E. Bixby, Implementing the simplex method: the initial basis, ORSA J. Comput. 4 (1992) 267–284.
[27] H.W. Kuhn, R.E. Quandt, An experimental study of the simplex method, in: E.A. Metropolis (Ed.), Proceedings of the 15th Symposium on Applied Mathematics, Amer. Math. Soc., 1963, pp. 107–124.
[28] N.K. Karmarkar, A new polynomial-time algorithm for linear programming, Combinatorica 4 (4) (1984) 373–395.
[29] J.E. Mitchell, P.M. Pardalos, M.G.C. Resende, Interior point methods for combinatorial optimization, in: D.-Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, Kluwer Academic Publishers, 1998, pp. 189–297.
[30] N.K. Karmarkar, K.G. Ramakrishnan, Computational results of an interior point algorithm for large scale linear programming, Math. Program. 52 (1991) 555–586.
[31] M. Iverson, F. Ozguner, G. Follen, Parallelizing existing applications in a distributed heterogeneous environment, in: Proceedings of Heterogeneous Computing Workshop, 1995, pp. 93–100.
[32] Montage. <http://montage.ipac.caltech.edu/> (accessed at 5.11.2007).
[33] M. Doar, I.M. Leslie, How bad is naive multicast routing? in: Proceedings of IEEE INFOCOM'93, 1993, pp. 82–89.
[34] O. Sinnen, L.A. Sousa, Communication contention in task scheduling, IEEE Trans. Parallel Distribut. Syst. 16 (6) (2005) 503–515.
[35] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05), vol. 2, 2005, pp. 759–767.
[36] X. He, X. Sun, G. von Laszewski, QoS guided min–min heuristic for grid task scheduling, J. Comput. Sci. Technol. 18 (4) (2003) 442–451.
[37] N. Fujimoto, K. Hagihara, Near-optimal dynamic task scheduling of precedence constrained coarse-grained tasks onto a computational grid, in: Proceedings of the Second International Symposium on Parallel and Distributed Computing, 2003, pp. 80–87.

Daniel Batista received a B.Sc. degree in Computer Science from Federal University of Bahia in 2004 and his M.Sc. degree in Computer Science from State University of Campinas in June 2006. He is now a Ph.D. student at the Institute of Computing, State University of Campinas, Campinas, Brazil and he is affiliated with the Computer Networks Laboratory at the same University. His research interests include traffic engineering and grid networks. His current research addresses optical networks mechanisms for grids.

Nelson L.S. da Fonseca received his Electrical Engineer (1984) and M.Sc. in Computer Science (1987) degrees from The Pontificial Catholic University at Rio de Janeiro, Brazil, and the M.Sc. (1993) and Ph.D. (1994) degrees in Computer Engineering from The University of Southern California, USA. Since 1995, he has been affiliated with the Institute of Computing of The State University of Campinas, Campinas – Brazil where is currently an Associate Professor. He is also a Consulting Professor to the Department of Informatics and Telecommunications of the University of Trento, Italy. He is the Editor-in-Chief of the IEEE Communications Surveys and Tutorials. He served as Editor-in-Chief of the IEEE Communications Society Electronic Newsletter and Editor of the Global Communications Newsletter. He is member of the editorial board of: Computer Networks, IEEE Communications Magazine, IEEE Communications Surveys and Tutorials, and Brazilian Journal of Computer Science. He served on the editorial board of the IEEE Transactions on Multimedia and on the board of the Brazilaina Journal on Telecommunications. He is the recipient of Elsevier Computer Networks Editor of the Year 2001, USC International Book award and of the Brazilian Computing Society First Thesis and Dissertations award. He is an active member of the IEEE Communications Society. He served as ComSoc Director of On-line Services (2002–2003) and served as technical chair for several ComSoc symposia and workshops. His main interests are traffic control, and multimedia services.

Flávio K. Miyazawa joined the Institute of Computing, State University of Campinas in 1998 and is currently an Associate Professor. He obtained his B.Sc. degree in Computer Science (1990) from the Federal University of Mato Grosso do Sul and the M.Sc. (1993) and Ph.D. (1997) in Applied Mathematics from the University of São Paulo. His main research activities are in the field of combinatorial optimization, mainly in network design, cutting stock and packing problems.

Fabrizio Granelli was born in Genoa in 1972. He received the "Laurea" (M.Sc.) degree in Electronic Engineering from the University of Genoa, Italy, in 1997, with a thesis on video coding, awarded with the TELECOM Italy prize, and the Ph.D. in Telecommunications from the same university, in 2001. Since 2000 he is carrying on his teaching activity as Assistant Professor in Telecommunications at the Department of Information and Communication Technology – University of Trento (Italy). In August 2004, he was visiting professor at the State University of Campinas (Brasil). He is author or co-author of more than 40 papers published in international journals, books and conferences, and he is member of the Technical Committee of the International Conference on Communications (ICC2003, ICC2004 and ICC2005) and Global Telecommunications Conference (GLOBECOM2003 and GLOBECOM2004). He is guest-editor of ACM Journal on Mobile Networks and Applications, special issue on "WLAN Optimization at the MAC and Network Levels" and Co-Chair of 10th IEEE Workshop on Computer-Aided Modeling, Analysis, and Design of Communication Links and Networks (CAMAD'04). He is General Vice-Chair of the First International Conference on Wireless Internet (WICON'05) and General Chair of the 11th IEEE Workshop on Computer-Aided Modeling, Analysis, and Design of Communication Links and Networks (CAMAD'06).

His main research activities are in the field of networking and signal processing, with particular reference to network performance modeling, medium access control, wireless networks, next-generation IP, and video transmission over packet networks.

He is Senior Member of IEEE and Associate Editor of IEEE Communications Letters.