Multi-Layer Traffic Engineering for Grid Networks*

Daniel M. Batista
State University of Campinas, Institute of Computing,
Campinas, SP – Brazil, CEP: 13084–971, CxP: 6176
batista@ic.unicamp.br

Abstract

This paper introduces a procedure for enabling grid applications to self-adjust their resource allocation. The procedure is based on monitoring the state of the available resources and on code migration of applications composed of dependent tasks. Moreover, a set of schedulers for applications with time requirements is presented and their performance evaluated.

Keywords: computer networks, computational grids, traffic engineering, task scheduling, task migration

*Dissertation [3] available at http://www.ic.unicamp.br/~batista/dissertacao-final.pdf

1 Introduction

Grid Networks (Grids) were designed to provide a distributed computational infrastructure for advanced science and engineering [14] [11]. 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.

Central to grid processing is the scheduling of application tasks to resources. The scheduling problem is an NP-hard problem [22], and feasible solutions in real time require either heuristics or approximations. Moreover, the computational complexity increases due to the need to account for heterogeneous resources and irregular topologies, in contrast to what happens in multiprocessor systems.

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 network resources is fundamental for the autonomous operation of a grid.

Adaptive scheduling and dynamic scheduling are two different approaches which try to adapt requirements of an application to the available resources. Adaptive scheduling deals with unforeseen resource demands at the scheduling time whereas dynamic scheduling tries to ameliorate the impact of fluctuations on resource availability by not scheduling at once all the tasks composing an application. In the latter approach, decisions about resource allocation to a task are postponed to the moment at which its dependencies are resolved. However, both adaptive scheduling and dynamic scheduling do not address three important issues: i) the production of feasible schedules in a reasonable amount of time when compared to the application execution time; ii) the impact of network links availability on the execution time of an application; and iii) the necessity to migrate code for decreasing the execution time of an application.

To overcome these challenges, this paper proposes a procedure for enabling grid applications composed by dependent tasks to adapt themselves to resource availability. This procedure involves task scheduling, resource monitoring and task migration, and its goal is to decrease the execution time of grid applications. The procedure involves the concept of self-adjustment as in Traffic Engineering for networks [2].

The procedure for self-adjustment differs from other approaches in the literature [1] [18] [28] by considering the fluctuation of bandwidth availability. It is most appropriate to applications composed of dependent tasks with a huge demand of data transfer, which is typical of e-science applications.

An additional contribution of this paper is a set of schedulers offering solutions which differ in terms of their schedule length as well as computational complexity. The distinguished aspect of this set of schedulers is the consideration of time requirements to produce feasible schedules. Their 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 shows numerical examples. Section 5 discusses related works and Section 6 provides 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. Figure 1 [25] illustrates the phases in the execution of a grid application. The bottom of the left side of Figure 1 illustrates the steps needed for scheduling.

Determination of resource availability and application needs constitutes the first phase of the process. The main question in scheduling is how to map the tasks of an application onto resources so that the execution time of the application, called schedule length, is minimized. The procedure introduced in this paper considers applications whose tasks can be described as Direct Acyclic Graphs (DAGs) in which 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

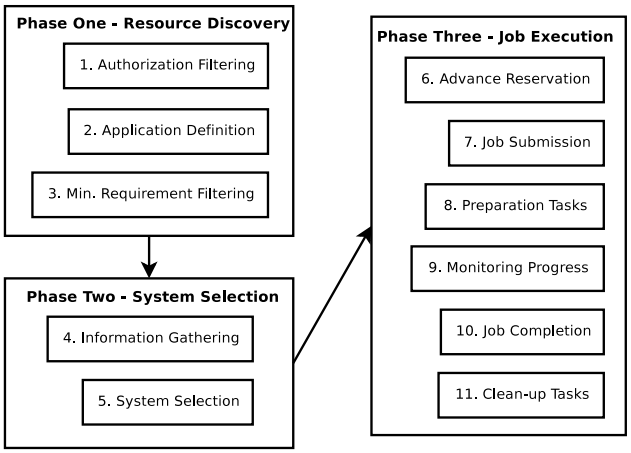


Figure 1: A grid scheduling process

of processing required for a task. Several e-science applications, such as those in astronomy and simulation of molecular dynamics, can be represented with DAGs. Figure 2 illustrates the DAG of a visualization application (remote rendering) [23] that will be used to illustrate the procedure for self-adjustment.

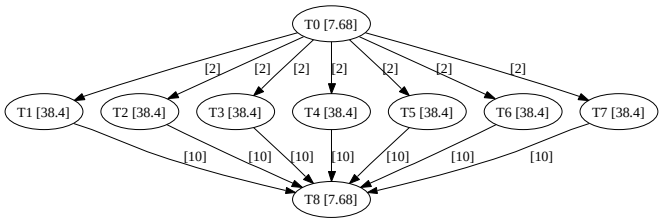


Figure 2: A grid application DAG

In this paper, CPU and bandwidth demands are considered. Other demands are not taking into account. However, this does not limit the contribution of the paper, yet make it easier to illustrate.

After the tasks are allocated to hosts (grid nodes) according to a schedule, tasks are executed until all have been terminated. However, due to the lack of ownership of resources, their 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 should be redefined so that a shorter schedule than those originally predicted can be achieved. Indeed, the procedure for self-adjustment enables grid applications to adapt themselves to current resource availability [4] [5] [8].

In order to provide this capability, it is necessary to monitor the network resources periodically and perform code migration accordingly. Code migration aims at reducing the execution time of a single application. The target is not the overall optimization of the utilization of the grid resources. The benefit of potential migrations is always balanced with the overhead to realize it. The cost of migration is accounted in all potential rescheduling of tasks. The accountability of code 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 network resources. Produce a schedule for the beginning of task execution and data transfer;
- *Step 2* Migrate the task codes and data to the hosts where the tasks will run. The execution of the tasks begins as soon as migration 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 scheduling of tasks. If no change is detected, continue monitoring the grid periodically (Step 3);
- *Step 5* Derive a new DAG representing current computation and data transfer demand and produce a schedule for these tasks;
- *Step 6* Check whether the schedule derived is equal to 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.

Figure 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) demands efficient schedulers. Section 3 will introduce eight novel schedulers for dealing with heterogeneous resources in a grid [7] [6].

In Steps 2 and 8, code and data migration can be performed using existing protocols, such as FTP and GridFTP [10]. It is assumed in Step 8 that it is possible to resume the execution of an interrupted task. One method for the resume of task execution is provided in [24]. Techniques for monitoring the available bandwidth [20] as well for predicting the network capacity with low computational overhead are available [29] [27] and can be used in Step 3.

The same schedulers used for the initial scheduling of the Application (Step 1) can be used for the rescheduling and migration of tasks due to changes in resource availability (Steps 5, 6 and 7). Rescheduling decisions consider resource availability and current execution status besides the initial schedule. Algorithm 1 implements Steps 5 to 7 and it uses the same scheduler used in Step 1.

Algorithm 1 Tasks rescheduling and migration

Input: Previous schedule; DAG with set of tasks $\mathcal{J}$; Description of current resource availability status; Current time; Scheduler.

1: **for** each task $i \in \mathcal{J}$ 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 of i.
4: Move all the outgoing arcs of i to i'.
5: Create an arc ii' with weight equivalent to the amount of bytes that need to be transferred in case task i migrates.
6: Assign to the variable h the id of the host to which the task i was mapped previously to the rescheduling decision.
7: Create a new constraint to Scheduler to force task i to be scheduled to h.
8: **end for**
9: **for** each task $k \in \mathcal{J}$ which either has already completed execution or is presently receiving data from others tasks **do**
10: Assign to the variable h the id of the host to which the task k was mapped previously to the rescheduling decision.
11: Create a new constraint to Scheduler to force task k to be scheduled to h.
12: **end for**
13: Execute the Scheduler with the new constraints and the new DAG.
14: **for** each task $i \in \mathcal{J}$ in execution **do**
15: **if** host to which i' be mapped $\neq$ host to which i was mapped previously to the rescheduling decision **then**
16: Migrate task i to the new host.
17: **end if**
18: **end for**

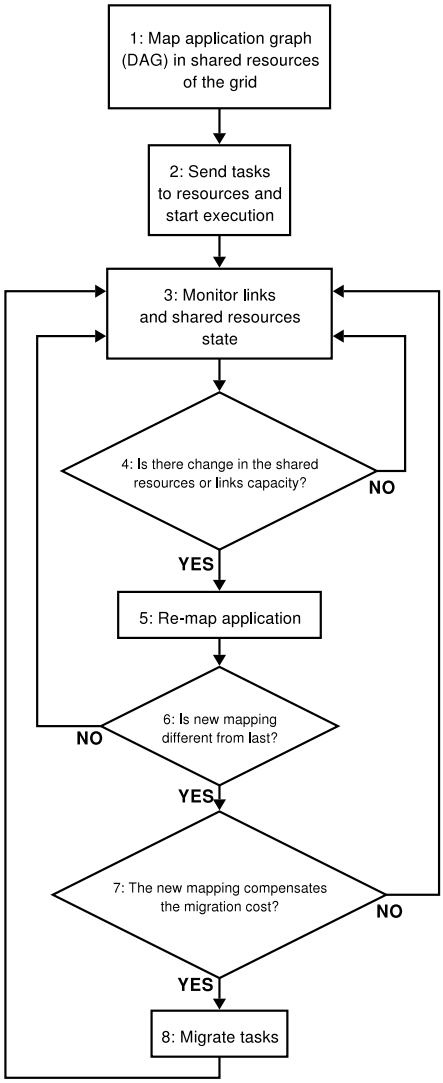


Figure 3: A Flow Diagram of the Procedure for Self-Adjustment of Resource Allocation

Algorithm 1 works on a modified DAGs of an application which represent the evolution of the execution up to a certain time. For each task i in execution, a new task i' represent the current execution status. Tasks that have already finished their execution are kept at the node they finished. Tasks receiving data from tasks they depend on are also kept in the same node. Task i will migrate only if task i' is mapped to a different resource than the one task i is mapped.

The self-adjusting capacity allows great flexibility and can be introduced in middlewares for grids such as [1] [18] [28]. Figure 4 illustrates the introduction of the procedure for self-adjustment into the scheme proposed in [25] which is represented on both sides of the figure. Note that according to the procedure in [25], once a task is scheduled to a host, it is executed until completion regardless of the fluctuation of resources availability. The central part of the figure is the procedure introduced here and it replaces the dashed part of the scheme in [25].

The procedure for self-adjustment can be considered a multilayer one. In [15], a layered architecture for grid networks was proposed. Figure 5 illustrates the mapping of this architecture to the layers of that of the Internet. The Resource layer serves as an intermediary between the application and the infrastructure of the grid. One of its main functionalities is the evaluation of application requirements as well as the gathering of information about the status of shared resources. The Collective layer is responsible for the selection of resources to meet the application requirements, as well as the allocation of these resources. The combination of these two layers corresponds to the Application layer in Internet architecture. The Connectivity layer,

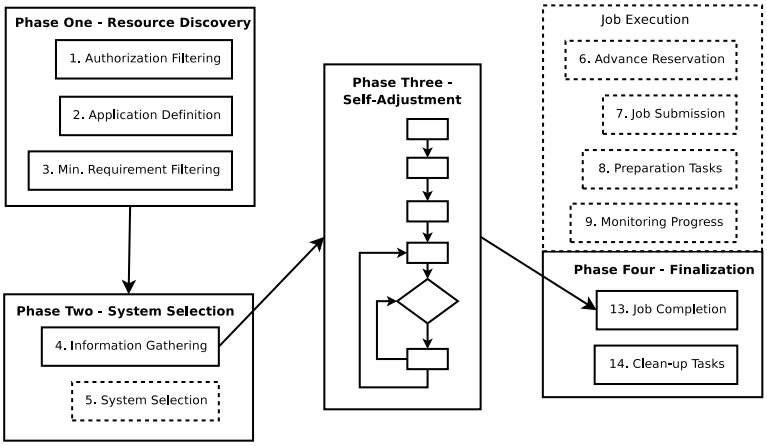


Figure 4: Inclusion of Procedure for Self-Adjustment in the process shown in Figure 1

which houses protocols for communication and data transfer, corresponds to a combination of the Transport and Internet layers of the architecture of the Internet. The Fabric layer is then responsible for communication at the link level.

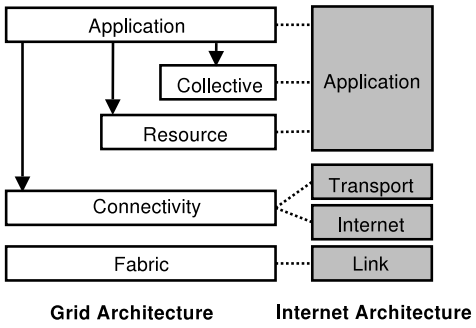


Figure 5: Relationship between Grid and Internet architectures

Table 1 shows that the procedure for self-adjustment involves three layers in the Grid Architecture, that, in turn, are mapped to three layers of the Internet Architecture.

Step	Grid Architecture Layer	Internet Architecture Layer
Monitoring – Steps 3 and 4	Collective Resource	Application
Rescheduling – Steps 5 to 7	Collective Resource	Application
Migration – Step 8	Connectivity	Transport Internet

Table 1: Relationship between steps and layers

3 Grid Schedulers

The problem of scheduling tasks to heterogeneous resources is a well-known NP-hard problem, and various sub-optimal solutions that can be achieved 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 produced under time requirements. Schedulers which execute fast can be employed

in Step 1 whereas those which gives schedules closer to the optimum one can be used in Steps 5 through 7, since those steps usually involves 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 anyone time.

The schedules produced by six of the eight schedulers proposed are derived from the solution of mixed/integer linear programming (LP) 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 mixed/integer programming problem for both continuous and discrete time are derived and the other four schedulers are derived by employing two different relaxation techniques to the exact LP problems.

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

Although solving exact linear 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 i^{th} task being mapped to the k^{th} host. Two different randomized rounding techniques were adopted to define two different algorithms. Algorithm 2 solves a linear programming 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.

Algorithm 2 Randomized Rounding

Input: Relaxation of Linear Program LP 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 fractional optimum solution of LP, where X = (X_{i,k}).
2: for P times do
3:    for each task i ∈ 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 on which it depends.
8:    Keep this schedule if it is the shortest one.
9: end for
10: Return the shortest schedule.
```

The other two schedulers are based on random drawing. The schedule is the 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

Algorithm 3 Iterative Randomized Rounding

Input: Relaxation of Linear Program LP to schedule the set of tasks $\mathcal{J}$ in the set of hosts $\mathcal{H}$; Q=Number of iterations.
Output: Schedule of $\mathcal{J}$ in $\mathcal{H}$.

```
for Q times do
    Let LP be the original linear program given in the input.
    Let X be the fractional optimum solution of LP, where X = (X_{i,k}).
    for each task i ∈ J do
        Let the probability of mapping the task i to the host k be X_{i,k}.
        Select a host to execute the task based on the mapping probability value.
        Add to the LP the constraint that the task i must be mapped to the host k.
        Let X be the fractional optimum solution of this new LP.
    end for
    Keep this schedule if it is the shortest one.
end for
Return the shortest schedule.
```

among the hosts, whereas in the other, it somehow translates the characteristics of tasks and hosts, 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 [21]. Some characteristics of the DAGs are:

- n: number of tasks ($n \in \mathbb{N}$);
- I_i: processing demand of the i^{th} 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 i^{th} task and the j^{th} task ($B_{i,j} \in \mathbb{R}_+$);
- $\mathcal{D}$: 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 k^{th} host takes to execute 1 instruction ($TI_k \in \mathbb{R}_+$);
- $TB_{k,l}$: time for transmitting 1 bit on the link connecting the k^{th} host and the l^{th} host ($TB_{k,l} \in \mathbb{R}_+$);
- $\mathcal{N}$: set $\{kl :$ host k is linked to host $l\}$. In particular, $kk \in \mathcal{N}$ for any host k and if $kl \in \mathcal{N}$ then we also have $lk \in \mathcal{N}$;
- $\delta(k)$: set of hosts linked to the k^{th} 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., $T_{max} = (\min TI) \sum_{i=1}^{n} I_i$, where $\min TI$ is the lowest value of TI. $\mathcal{J} = \{1, \ldots, n\}$ is the set of existing tasks of an application and $\mathcal{H} = \{1, \ldots, m\}$ is the set of hosts.

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

3.1 LP Formulation with Time as a Continuous Variable

This approach adopts a mixed linear programming formulation for the grid scheduling problem:

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

such that

$$s_i \geq s_0 \qquad \text{for } i \in \mathcal{J}; \qquad \text{(C1)}$$

$$s_j \geq s_i + \sum_{k \in \mathcal{H}} [(I_i TI_k X_{i,k}) + \sum_{l \in \delta(k)} (B_{i,j} TB_{k,l} VA_{i,k,j,l})] \qquad \text{for } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}; \qquad \text{(C2)}$$

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

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

$$\sum_{k \in \mathcal{H}} X_{i,k} = 1 \qquad \text{for } i \in \mathcal{J}; \qquad \text{(C5)}$$

$$\sum_{k \in \mathcal{H}} \sum_{l \in \delta(k)} VA_{i,k,j,l} = 1 \qquad \text{for } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}; \qquad \text{(C6)}$$

$$2\,VA_{i,k,j,l} \leq X_{i,k} + X_{j,l} \qquad \text{for } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad k,l \in \mathcal{H}, \quad kl \in \mathcal{N}; \qquad \text{(C7)}$$

$$VA_{i,k,j,l} - X_{i,k} - X_{j,l} \geq -1 \qquad \text{for } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad k,l \in \mathcal{H}, \quad kl \in \mathcal{N}; \qquad \text{(C8)}$$

$$2\,VA_{i,k,j,k} \leq X_{i,k} + X_{j,k} \qquad \text{for } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}, \quad k \in \mathcal{H}; \qquad \text{(C9)}$$

$$VA_{i,k,j,k} - X_{i,k} - X_{j,k} \geq -1 \qquad \text{for } i,j \in \mathcal{J}, \quad i \neq j, \quad ij \notin \mathcal{D}, \quad ji \notin \mathcal{D}, \quad k \in \mathcal{H}; \qquad \text{(C10)}$$

$$VA_{i,k,j,l}, X_{i,k}, P_{i,j} \in \{0,1\} \qquad \text{for } i,j \in \mathcal{J}, \quad k,l \in \mathcal{H}. \qquad \text{(C11)}$$

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 i^{th} task is executed first (in which case constraint (C4) is satisfied) and 0 if the j^{th} task is executed first (constraint (C3) is satisfied). 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 i^{th} and j^{th} tasks are scheduled to the k^{th} and to the l^{th} hosts, respectively.

Constraints (C7), (C8), (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.

The final scheduling is established by the value of the following variables:

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

This formulation must account for a maximum of $(\frac{m^2}{2} + \frac{3m}{2} + 3)n^2 - (\frac{m^2}{2} + \frac{3m}{2} + 1)n$ constraints, and $(m^2 + 1)n^2 + (m + 1)n$ variables. The scheduler based on the exact solution of this problem involving mixed linear programming with a continuous time variable is denominated MLPCT. There are two more versions of MLPCT, one involving Algorithm 2 based on randomized rounding (CT-RR) and the other using Algorithm 3 based on iterative randomized rounding (CT-IRR).

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

3.2 LP Formulation with Time as a Discrete Variable

This formulation considers discrete intervals of time and treats the scheduling problem as an integer linear programming problem, and is formulated as follows:

Minimize f_n

such that

$$\sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{H}} x_{j,t,k} = 1 \qquad \text{for } j \in \mathcal{J}; \qquad \text{(D1)}$$

$$f_j = \sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{H}} t\, x_{j,t,k} \qquad \text{for } j \in \mathcal{J}; \qquad \text{(D2)}$$

$$x_{j,t,k} = 0 \qquad \text{for } j \in \mathcal{J}, \quad k \in \mathcal{H}, \quad t \in \{1, \ldots, \lceil I_j\, TI_k \rceil\}; \qquad \text{(D3)}$$

$$\sum_{k \in \delta(l)} \sum_{s=1}^{\lceil t - I_j\, TI_l - B_{i,j}\, TB_{k,l} \rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l} \qquad \text{for } i,j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad \text{for } l \in \mathcal{H}, \quad t \in \mathcal{T}; \qquad \text{(D4)}$$

$$\sum_{j \in \mathcal{J}} \sum_{s=t}^{\lceil t + I_j\, TI_k - 1 \rceil} x_{j,s,k} \leq 1 \qquad \text{for } k \in \mathcal{H}, \quad t \in \mathcal{T}, \quad t \leq \lceil T_{max} - I_j\, TI_k \rceil; \qquad \text{(D5)}$$

$$\sum_{l \in \mathcal{H}} VA_{j,l} = 1 \qquad \text{for } j \in \{2, \ldots, n\}; \qquad \text{(D6)}$$

$$(|\{i : ij \in \mathcal{D}\}| + 1)\, VA_{j,l} \leq \sum_{t \in \mathcal{T}} x_{j,t,l} + \sum_{i : ij \in \mathcal{D}} \sum_{k \in \delta(l)} \sum_{t \in \mathcal{T}} x_{i,t,k} \qquad \text{for } j \in \{2, \ldots, n\}, \quad l \in \mathcal{H}; \qquad \text{(D7)}$$

$$|\{i : ij \in \mathcal{D}\}| + VA_{j,l} \geq \sum_{t \in \mathcal{T}} x_{j,t,l} + \sum_{i : ij \in \mathcal{D}} \sum_{k \in \delta(l)} \sum_{t \in \mathcal{T}} x_{i,t,k} \qquad \text{for } j \in \{2, \ldots, n\}, \quad l \in \mathcal{H}; \qquad \text{(D8)}$$

$$VA_{j,l},\ x_{j,t,l} \in \{0,1\} \quad \text{for } j \in \mathcal{J}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}. \quad \text{(D9)}$$

for convenience, the following notations are used: $\mathcal{T} = \{1, \ldots, T_{max}\}$ and $X_{i,k}$ is defined as $\sum_{t=1}^{T_{max}} x_{i,k,t}$.

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 i^{th} task finished at time t in the host k; otherwise this variable assumes a value of 0;
- f_i: Variable that stores the time at which the execution of the i^{th} task is finished ($f_i \in \mathbb{N}^*$).

The relaxation of the discrete time formulation consists of changing the set $\{0, 1\}$ of the constraints in (D9) 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) establish the finishing time for all the tasks, and the constraints in (D3) determine that a task (j) cannot terminate until it has been executed in the host k. The constraints in (D4) establish that if the i^{th} task executes in the l^{th} host before the j^{th} task does, and that the j^{th} task is finished at time t, then the time when the i^{th} task finished its execution is at most t minus the execution time of the j^{th} task minus the time needed to transfer data between these two tasks. The constraints in (D5) establish that there is at most one task in execution at any one host at a specific time, while the constraints in (D6) guarantee that a task be scheduled to a single host at any one time, while the constraints in (D7) and (D8) determine that dependent tasks are mapped to hosts interconnected.

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 a maximum of $(n^2 + n + 2)\frac{m}{2}T_{max} + (2n - 2)m + 3n - 1$ constraints and $(mT_{max} + m + 1)n$ variables. The scheduler based on an exact solution of the integer linear programming with a discrete time variable is denominated as ILPDT. 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 a 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.

Algorithm 4 Random Drawing with Uniform Probability Distribution

Input: DAG with set of tasks $\mathcal{J}$; Description of current resource availability status $\mathcal{H}$; P=Number of drawings.
Output: Schedule of $\mathcal{J}$ in $\mathcal{H}$

1: **for** P times **do**
2: Set the probability value for scheduling each task to a host as $1/m$.
3: **for** each task $i \in \mathcal{J}$ **do**
4: Assign randomly a host $k \in \mathcal{H}$ 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.

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:

- The probability that a task will be executed in a given host is proportional to the processing rate of all available hosts;
- 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;
- 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;
- The larger the number of edges in a DAG, the higher is the probability that a given task will be assigned to a host with large number of links connecting it to other hosts;
- 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;
- 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.

This set of rules is denominated "set of rules 1" in Algorithm 5. The first two rules define the initial probability of mapping the i^{th} task to the k^{th} host, given by:

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

If the criteria used were limited to grid resources, hosts with greater availability of processing rates and bandwidths would be utilized all the time, whereas hosts with less capacity would be 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 [19] [21]. Consequently, the probability value in Equation 1 is redefined to each task considering the last four rules defined above.

This DG scheduler is presented in Algorithm 5.

Algorithm 5 Drawing using Distribution involving "Grid-Aware" Probabilities Values

Input: DAG with set of tasks $\mathcal{J}$; Description of current resource availability status $\mathcal{H}$; P=Number of drawings.
Output: Schedule of $\mathcal{J}$ in $\mathcal{H}$

```
for P times do
    Set the probability for scheduling a task to a host on the basis of the "set of rules 1".
    Redefine the probabilities on the basis of the last four rules.
    for each task i ∈ J do
        Select randomly the host k ∈ H for the execution of the i^th task.
        Normalize the probability values of the tasks dependent on the i^th task.
        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.
    end for
    Consider this schedule if it produces the shortest execution time so far.
end for
Return the schedule with the shortest schedule.
```

3.5 Comparison of Schedulers Efficiency

Various network topologies and tasks DAGs were used to compare schedulers proposed here. Results of the experiments involving the DAG shown in Figure 6 are representative of those obtained in other experiments and will be presented in this section. 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 2GB RAM was used in the experiments. The software `Xpress`[12] was employed to solve the linear programming problems. Computer programs were developed using the C language.

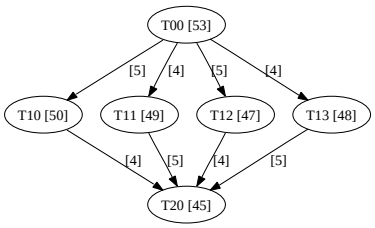


Figure 6: Tasks DAG used in the experiments

Various topologies were generated using the Doar-Leslie method [13] by changing the number of hosts, the network connectivity (vertex degree) and the ratio between the longest and the shortest edge. 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 long edges and the number of short ones ($\in (0, 1]$) 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 means bandwidth availability. Values of connectivity close to 1 gives complete graphs.

If not stated otherwise, 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. The weights of the DAG arcs in Figure 6 were in the interval $[4, 5]$, whereas the weight of the vertices varied in the interval $[45, 53]$. Furthermore, excepting 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 time were experienced with other values.

For schedulers which considers time as a discrete variable, it is advisable to use a discretizing value which corresponds to a fraction of the serial execution time of the DAG. Preliminary experiments indicate that 6.25% is a good choice, corresponding to a time unity of 8 minutes.

In the evaluation of the schedulers, their execution time and speedup as a function of the number of nodes, network connectivity and the edge weights are compared.

Tables 2 and 3 show the results when the number of nodes (hosts) is varied. Table 2 presents the performance of the proposed schedulers as a function of the number of hosts. The performance of MLPCT 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, MLPCT took over one hour to generate a schedule, whereas ILPDT took 12.3 seconds. 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. ILPDT produced the largest speedup for most of the experiments, followed by CT-RR. Schedulers based on the relaxation in Algorithm 3 (CT-IRR and DT-IRR) produced the smallest speedup among the schedulers based on linear 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 since the initial probability values of the former consider both grid and task constraints.

Hosts	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
10	77.72%	77.72%	99.31%	77.55%	98.51%	99.07%	**1.289432**
40	89.21%	73.36%	99.86%	73.26%	**1.365060**	81.79%	85.10%
70	**1.556116**	64.34%	91.51%	82.48%	99.38%	77.25%	84.52%
100	**1.534463**	65.55%	97.73%	65.17%	94.18%	70.73%	79.10%
130	94.38%	66.71%	91.73%	66.23%	**1.509969**	69.67%	75.53%
160	93.43%	62.23%	91.33%	62.11%	**1.610028**	68.26%	73.84%
190	62.49%	62.49%	81.82%	62.27%	**1.606021**	65.29%	74.71%

Table 2: Speedup as a function of the number of hosts

The execution time of schedulers based on linear programming, portrayed in Table 3, increases as the number of hosts increases with the largest speedups achieved by schedulers which took longer to generate the schedule. Clearly, this does not happens with schedulers based on random drawing. The use of algorithms

Hosts	Execution time (seconds)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
10	0.28	0.05	0.52	0.21	0.17	0.08	**0.07**
40	3.64	0.64	1.63	2.00	**12.30**	0.60	0.48
70	**16.87**	2.47	5.14	5.19	4.38	1.80	1.38
100	**78.02**	7.32	10.02	9.55	17.80	3.40	2.56
130	71.12	19.92	17.29	23.96	**81.31**	5.86	4.49
160	119.98	55.83	26.16	33.18	**24.85**	8.56	6.54
190	345.77	82.04	25.64	32.05	**134.03**	11.89	8.98

Table 3: Execution time as a function of the number of hosts

based on integrality relaxation led to lower execution times than did their exact counterparts. For a 190-host network, ILPDT took 134.03 seconds while DT-RR took 25.64 seconds. Moreover, the execution time of schedulers based on integrality relaxation does not increase as fast as the exact ones do. Discretization of time plays a key role in decreasing the execution time as can be seen in the comparison of the time demanded by CT-RR with that required by its discrete time counterpart (DT-RR).

Table 4 shows the speedup and Table 5 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 Table 2 and Table 3, ILPDT and DT-RR generally produce the best schedules. DG did generated two 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. The largest speed up is no longer associated with the longest execution time, as happened when the number of hosts was varied.

Conect.	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
0.1	71.06%	71.06%	84.72%	81.08%	83.81%	96.38%	**1.408825**
0.22	72.97%	72.97%	96.23%	72.55%	96.46%	89.29%	**1.378274**
0.34	70.09%	69.64%	91.51%	69.64%	**1.435858**	97.30%	98.64%
0.46	98.17%	66.10%	**1.517342**	65.90%	97.64%	81.37%	92.93%
0.58	89.92%	66.10%	**1.522505**	69.92%	99.64%	89.51%	92.97%
0.7	98.82%	65.32%	98.82%	65.31%	**1.531184**	77.78%	90.87%
0.82	91.71%	66.10%	**1.524155**	77.74%	65.61%	73.64%	87.03%

Table 4: Speedup as a function of network connectivity

Conect.	Execution time (seconds)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
0.1	0.82	0.56	1.69	0.86	18.84	1.38	**1.10**
0.22	5.97	0.71	1.66	1.33	12.21	1.38	**1.09**
0.34	4.56	1.24	2.33	1.41	**32.28**	1.26	0.99
0.46	4.01	1.24	**3.53**	3.08	7.49	1.08	0.81
0.58	4.85	0.93	**2.35**	1.77	2.22	0.96	0.73
0.7	7.30	1.73	3.04	3.69	**3.46**	0.38	0.30
0.82	9.98	1.24	**2.60**	3.05	4.29	0.10	0.09

Table 5: Execution time as a function of network connectivity

Table 6 and Table 7 shows the performance of the schedulers as a function of the ratio between the longest and the shortest edge. The use of ILPDT led to largest speedups but to the longest execution time, although there is no clear pattern involving an increase in execution time as a function of the ratio between the longest and the shortest edge.

From the results found in those experiments, the scheduler which generated the largest speed up was the ILPDT but at the cost of execution time. Schedulers based on Algorithm 2 produced results which were close to those given by ILPDT but the execution time required, especially for DT-RR was much less than that for ILPDT, and this savings justify eventual small losses in speedup. Whenever time requirements are too limited to wait for a schedule derived via linear programming the DG scheduler can also be used, since it produces reasonable speedup values under certain circumstances.

Ratio	Speedup						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
0.2	72.81%	72.81%	90.99%	72.42%	**1.380837**	77.84%	89.72%
0.3	92.53%	61.79%	92.53%	89.71%	**1.624240**	78.61%	84.50%
0.4	100.00%	68.55%	**1.469889**	81.06%	92.27%	86.55%	78.44%
0.5	74.89%	74.89%	92.55%	83.85%	**1.341922**	85.77%	90.06%
0.6	87.12%	64.61%	92.15%	86.03%	**1.552311**	75.85%	86.59%
0.7	68.58%	68.58%	82.75%	71.95%	**1.459533**	75.38%	82.35%
0.8	91.03%	68.17%	93.71%	67.72%	**1.476768**	83.85%	86.73%

Table 6: Speedup as a function of the ratio between longest and shortest edge

Ratio	Execution time (seconds)						
	CT-RR	CT-IRR	DT-RR	DT-IRR	ILPDT	RDU	DG
0.2	7.71	1.24	2.11	2.60	**13.64**	1.06	0.81
0.3	4.90	1.02	2.45	1.88	**9.75**	1.00	0.75
0.4	4.29	1.22	**2.26**	2.79	5.22	1.08	0.84
0.5	3.33	2.02	2.22	2.64	**3.56**	0.93	0.73
0.6	5.67	1.36	2.22	3.09	**2.10**	0.99	0.77
0.7	3.67	1.08	2.22	2.51	**3.28**	0.98	0.76
0.8	4.88	0.95	2.37	1.98	**29.85**	0.93	0.70

Table 7: Execution time as a function of the ratio between longest and shortest edge

4 Examples of the Use of the Procedure for Self-Adjustment

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

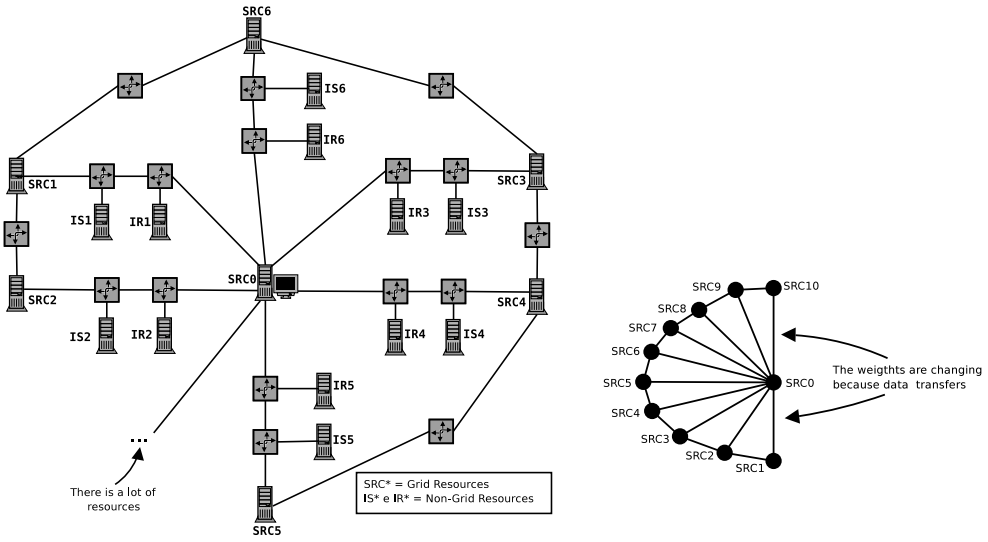
The application is the one described in Figure 2, whereas the grid is illustrated in Figure 7. The left hand side of Figure 7 shows the network topology whereas the right hand side, the grid nodes. The arc weights in the DAG represent the amount of data to transfer in GigaBytes, whereas the vertex weights represent the amount of instructions in a 10^{12} scale. The network has 34 hosts arranged around a central host, SRC_0, and the grid has 11 nodes, named $SRC_{\{0...10\}}$. The available processing rate of the host SRC_0 is 1600MIPS, whereas all others can process at the rate of 8000MIPS. The links connecting SRC_0 to the other hosts has the capacity of 100Mbps, whereas all the other links are limited to 33.33Mbps. The topology used resemble CERN's LHC Computing Grid. Note that the topology is not centralized around SRC_0 since the hosts can communicate without going through this node. Moreover, the processing capacity of this node is lower that those of the other hosts which implies in parallel execution of the tasks in other hosts.

Confidence intervals with 95% confidence level was derived via the batch means method. The width of the intervals was less than 5% of its mean value. Confidence intervals are omitted for the sake of visual interpretation.

The first experiment aims at finding the application execution time under ideal conditions so that it can be used for further comparison. In the second experiment, bandwidth is reduced and all the steps of the procedure for self-adjustment are executed. The third experiment includes the increase in resource availability and the last experiment evaluates the impact of the monitoring frequency on the performance.

In the first experiment, the application (Figure 2) is mapped using the MLPCT scheduler and the resulting mapping is $0 \rightarrow SRC_0$, $1 \rightarrow SRC_2$, $2 \rightarrow SRC_5$, $3 \rightarrow SRC_8$, $4 \rightarrow SRC_4$, $5 \rightarrow SRC_1$, $6 \rightarrow SRC_9$, $7 \rightarrow SRC_10$, $8 \rightarrow SRC_0$. Similar mapping could have been involved other hosts, since the topology is symmetrical. In the actual schedule derived, Tasks 1 to 7 start running at the time of 82.66min, whereas task number 8 starts running at 175.96min and finishes at 255.96min.

In the second experiment, the same scenario and initial mapping were used. However, at 90min, UDP streams with a rate of 90Mbps 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 minutes (The use of long monitoring intervals reinforce the effectiveness of monitoring the availability of resources). Thus, at the



(a) Network Topology (b) Virtual Organization Graph

Figure 7: Grid used in the examples

time 120 minutes, the need to re-evaluate 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 Figure 8.

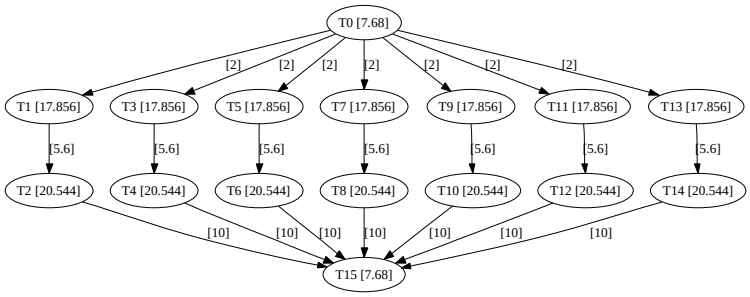
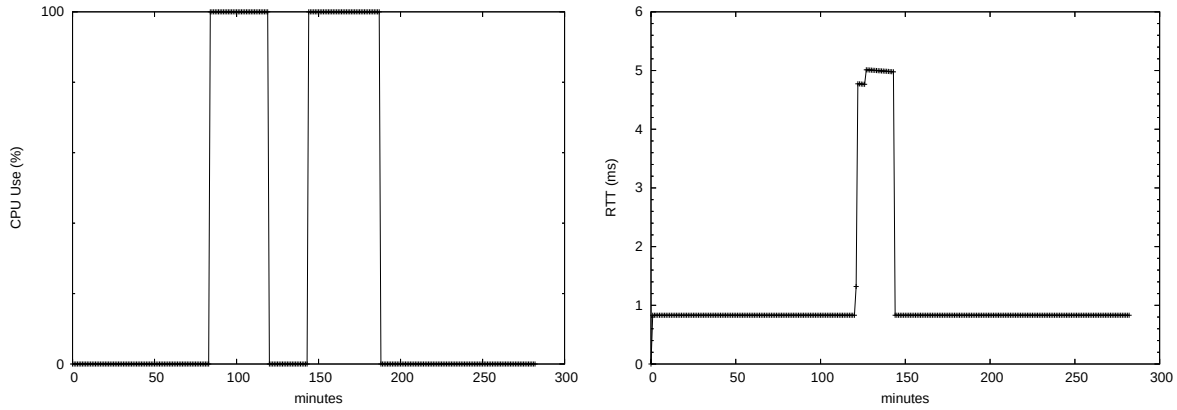


Figure 8: DAG for migration at 120min

For that DAG, the schedule was obtained by the ILPDT 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 10GB of data to Task 8. When migrations occur the new execution time was 281min, 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 358min, i.e., an increase of about 27.4%. Figures 9(a) and 9(b) 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 120min and 150min, no processing activity took place in either of the hosts SRC_2 and SRC_3, since in this intervals, migration take place.

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. Figure 10 illustrates the addition of the host SRC_{16}; the link capacity joining it to host SRC_6 is 1Gbps, with an available processing rate of 8000MIPS. Similar hosts were also added to hosts SRC_1 to SRC_{10}. With this extra resource, the execution time decreases to 247min, 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



(a) Use of CPU for Task 1

(b) RTT between SRC_2 and SRC_3

Figure 9: Task migration

available. If, for example, the processing rate available were 4000MIPS, migration would not be advisable since, execution time would have increased to 291min if migration were carried out, which is 13.23% higher than that obtained under ideal conditions.

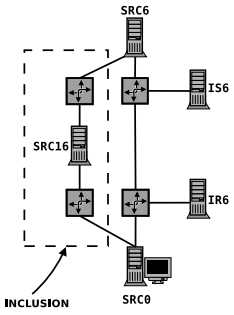


Figure 10: Inclusion of new resource linked to SRC_6

One of the key issues involved in the Self-Adjustment procedure is the frequency of re-evaluation 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 (60Mbps) was introduced to the same links as in the previous example. Both MLPCT and ILPDT schedulers were used for the experiments. First, a simulation with no task migration was run; execution time was 279min. Then, the recommendations of the scheduler were followed. Table 8 shows the execution time required when task migration is undertaken.

Interval	MLPCT	ILPDT
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)

Table 8: Execution times as a function of monitoring interval duration (minutes)

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 ILPDT may produce schedules which yield longer execution times than those where no migration is pursued, as can be in the results when intervals of 120min and 130min were used. Such imprecision is critical when approaching the "ideal" time for re-evaluation due to the approximations introduced by time discretization. In fact, the ideal frequency for re-evaluation 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) [29], 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) [27] 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 project was introduced in [20]; this adopts either active or passive monitoring techniques, depending on the network load. All these proposals for monitoring status of resources can be incorporated in Steps 3 and 4 of the Self-Adjustment procedure introduced in Section 2. However, the prediction of performance in Self-Adjustment procedure involves schedulers based on optimization for determining potential re-configuration of a grid.

Several self-adjusting systems based on monitoring and task migration have been proposed [1] [18] [28] in the literature. Although under different names, all these schemes aim at minimizing the execution time of the applications. In all these approaches, mechanisms are inserted in existing middlewares and agents for management of grid applications. The procedure for self-adjustment introduced in this paper differs from these schemes since it uses neither adaptive scheduling nor dynamic scheduling.

Although the proposal in [1] takes into account the decrease on the application performance, no evidence of the effectiveness of its policy for migration was presented. In this approach, an intermediate storage node can be used for migration to the final destination. In spite of the flexibility added by the intermediate node, this node can become a bottleneck.

The scheme in [18] promotes migration either if there is a disconnection in the grid or if there are changes in the application requirements. It uses a greedy algorithm for fast initial scheduling and adjust the schedule in succeeding evaluation steps.

In [28], migration occurs only if the gain in execution time is higher than 30%. The authors admit that this threshold value may not be the optimum one. However, the distinct difference between the self-adjustment procedure and the one in [28] is that the procedure for self-adjustment computes the migration overhead based on the current grid status whereas the proposed in [28] fixes the overhead estimation to a constant values.

Various scheduling schemes have been proposed for grids [21] [26] [9] [17] [16]. The Level-Branch Priority (LBP) [21] 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 [26] 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 [9]. 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 [17] 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 [16] 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 and Future Works

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 resources allocation approach for the empowerment of grids in this direction. The effectiveness of this new procedure has been illustrated in 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. In the future, the dynamic determination of the duration of intervals for re-evaluation of schedule needs to be pursued. Moreover, the resource allocation scheme proposed here shall be introduced into existing systems.

Acknowledgements

This work was supported in part by ProNEx–FAPESP/CNPq (Proc. 09850-5), CNPq (Proc. 131590/2004-9 and 305076/2003-5) and FAPESP Kyatera (Proc. 03/08277-0).

References

[1] Allen, G., Angulo, D., Foster, I., Lanfermann, G., Liu, C., Radke, T., Seidel, E., and Shalf, J. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *International Journal of High Performance Computing Applications 15*, 4 (Nov. 2001), 345–358.

[2] Awduche, D., Chiu, A., Elwalid, A., Widjaja, I., and Xiao, X. RFC 3272: Overview and Principles of Internet Traffic Engineering, 2002.

[3] Batista, D. M. Engenharia de Tráfego Multi-Camada para Grades. Master's thesis, Instituto de Computação – Unicamp, Jun 2006. `http://www.ic.unicamp.br/~batista/dissertacao-final.pdf`. Acessed at 30 Apr 2007.

[4] Batista, D. M., da Fonseca, N. L. S., Granelli, F., and Kliazovich, D. Self-Adjusting Grid Networks. In *Proceedings of the IEEE International Conference on Communications – ICC 2007 (Accepted for publication)* (Jun 2007).

[5] Batista, D. M., da Fonseca, N. L. S., Granelli, F., and Kliazovich, D. Uma Metodologia para o Auto-Ajuste de Aplicações em Grades. In *XXVII Congresso da Sociedade Brasileira de Computação – VI Wperformance (Submitted)* (Jul 2007).

[6] Batista, D. M., da Fonseca, N. L. S., and Miyazawa, F. K. Escalonadores de Tarefas em Grades. In *Anais do XXVI Congresso da Sociedade Brasileira de Computação – V Wperformance* (Jul 2006), pp. 73–92.

[7] Batista, D. M., da Fonseca, N. L. S., and Miyazawa, F. K. A Set of Schedulers for Grid Networks. In *SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing* (Mar 2007).

[8] Batista, D. M., da Fonseca, N. L. S., Miyazawa, F. K., and Granelli, F. Self-Adjustment of Resource Allocation for Grid Applications. *Computer Networks (In submission process)* (2007).

[9] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., and Kennedy, K. Task Scheduling Strategies for Workflow-based Applications in Grids. In *Proc. IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05)* (May 2005), vol. 2, pp. 759–767.

[10] Cannataro, M., Mastroianni, C., Talia, D., and Trunfio, P. Evaluating and Enhancing the Use of the GridFTP Protocol for Efficient Data Transfer on the Grid. *Lecture Notes in Computer Science 2840* (Jan 2003), 619 – 628.

[11] Casanova, H. Distributed Computing Research Issues in Grid Computing. *SIGACT News 33*, 3 (2002), 50–70.

[12] dash optimization. The Xpress-Optimizer. `http://www.dashoptimization.com/home/products/products_optimizer.html`. Acessed at 30 Apr 2007.

[13] Doar, M., and Leslie, I. M. How Bad is Naive Multicast Routing? In *Proc. IEEE INFOCOM'93* (1993), pp. 82–89.

[14] Foster, I. What is the Grid? A Three Point Checklist. *GRIDToday 1*, 6 (Jul 2002). `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`. Acessed at 30 Apr 2007.

[15] Foster, I., Kesselman, C., and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of High Performance Computing Applications 15*, 3 (Fall 2001), 200–222.

[16] Fujimoto, N., and Hagihara, K. Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid. In *Proc. Second International Symposium on Parallel and Distributed Computing* (Oct 2003), pp. 80–87.

[17] He, X., Sun, X., and von Laszewski, G. QoS Guided Min-Min Heuristic for Grid Task Scheduling. *Journal of Computer Science and Technology 18*, 4 (Jul 2003), 442–451.

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

[19] Iverson, M., Ozguner, F., and Follen, G. Parallelizing Existing Applications in a Distributed Heterogeneous Environment. In *Proc. Heterogeneous Computing Workshop* (Apr 1995), pp. 93–100.

[20] Lowekamp, B. B. Combining Active and Passive Network Measurements to Build Scalable Monitoring Systems on the Grid. *SIGMETRICS Performation Evaluation Review 30*, 4 (2003), 19–26.

[21] Ma, D., and Zhang, W. A Static Task Scheduling Algorithm in Grid Computing. *Lecture Notes in Computer Science 3033* (2004), 153–156.

[22] Papadimitriou, C. H., and Steiglitz, K. *Combinatorial Optimization – Algorithms and Complexity.* Dover Publications, 1998, pp. 363–366.

[23] Renambot, L., van der Schaaf, T., Bal, H. E., Germans, D., and Spoelder, H. J. W. Griz: Experience with Remote Visualization over an Optical Grid. *Future Generation Computer Systems 19*, 6 (2003), 871–882.

[24] Roy, A., and Livny, M. Condor and Preemptive Resume Scheduling. In *Grid Resource Management: State of the Art and Future Trends (1st Edition)* (2003), Springer, pp. 135–144.

[25] Schopf, J. M. Ten Actions when Grid Scheduling. In *Grid Resource Management: State of the Art and Future Trends (1st edition)* (2003), Springer, pp. 15–23.

[26] Sinnen, O., and Sousa, L. A. Communication Contention in Task Scheduling. *IEEE Transactions on Parallel and Distributed Systems 16*, 6 (Jun 2005), 503–515.

[27] Sun, X., and Wu, M. GHS: A Performance System of Grid Computing. In *Proc. 19th IEEE International Parallel and Distributed Processing Symposium* (Apr 2005), pp. 228a–228a.

[28] Vadhiyar, S. S., and Dongarra, J. J. A Performance Oriented Migration Framework for the Grid. In *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid(CCGRID'03)* (2003), pp. 130–137.

[29] Wolski, R., Spring, N. T., and Hayes, J. The Network Weather Service: a Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems 15*, 5–6 (1999), 757–768.