J Grid Computing (2008) 6:159–175
DOI 10.1007/s10723-006-9061-5

A Launch-time Scheduling Heuristics for Parallel Applications on Wide Area Grids

Ranieri Baraglia · Renato Ferrini · Nicola Tonellotto · Laura Ricci · Ramin Yahyapour

Received: 26 June 2006 / Accepted: 4 November 2006 / Published online: 17 March 2007

Abstract Large and dynamic computational Grids, generally known as wide-area Grids, are characterized by a large availability, heterogeneity on computational resources, and high variability on their status during the time. Such Grid infrastructures require appropriate schedule mechanisms in order to satisfy the application performance requirements (QoS). In this paper we propose a launch-time heuristics to schedule component-based parallel applications on such kind of Grid. The goal of the proposed heuristics is threefold: to meet the minimal task computational requirement, to maximize the throughput between communicating tasks, and to evaluate on-the-fly the resource availability to minimize the aging effect on the resources state. We evaluate the proposed heuristics by simulations applying it to a suite of task graphs and Grid platforms randomly generated. Moreover, a further test was conducted to schedule a real application on a real Grid. Experimental results shown that the proposed solution can be a viable one.

R. Baraglia · R. Ferrini · N. Tonellotto (✉)
Information Science and Technologies Institute, ISTI-CNR, Via G. Moruzzi 1, Pisa, Italy
e-mail: nicola.tonellotto@isti.cnr.it

L. Ricci
Department of Computer Science, University of Pisa, Largo B. Pontecorvo 3, Pisa, Italy

R. Yahyapour
Institute for Robotics Research, University of Dortmund, Otto-Hahn Straße 8, Dortmund, Germany

Keywords Grid computing · Wide-area Grids · Performance requirements · Static scheduling · Quality of service

1 Introduction

A Grid is a dynamic, seamless, integrated computational and collaborative environment. Grids integrate networking, computation and information to provide a virtual platform for computing and data management. Resources in a Grid are typically grouped into autonomous administrative domains communicating through high-speed communication links [17, 18].

A Grid Resource Management System (RMS) is fundamental for the operation of a Grid, and its basic functions are to accept requests for resources and assign specific resources to a request from the overall pool of Grid resources for which a user has access permission. The Grid RMS is composed by the middleware, tools and services allowing to disseminate resource information, to discover suitable resources and to schedule resources for job execution. The design of a Grid RMS must consider aspects such as:

site autonomy, heterogeneity, extensibility, allocation/co-allocation, scheduling, and online management [9, 21, 25, 28]. Regarding the scheduling policy adopted within a Grid RMS, we can distinguish between the system-oriented and application-oriented ones. Local job schedulers usually adopts system-oriented policies, trying to optimize system throughput. On the other hand, application schedulers try to optimize application-specific metrics like completion time.

Even if, in the broadest sense, any application can be a Grid application, in practice, applications that can take more advantage of using the Grid are loosely-coupled, computational intensive, component-based, and, often, multidisciplinary. The use of components provide suitable means to easily build efficient Grid applications [19, 29]. Components can be sequential or parallel, written using different programming languages, and may be distributed across the computational resources of the Grid.

Complex distributed application schedulers are needed to reach application goals such as desired application performance and scalability. The problem with such applications is that scheduling decisions taken at launch-time (static scheduling) cannot be sufficient to maintain the application performance requirements and need to be modified (dynamic re-scheduling) at run-time. Static scheduling is used before an application starts to determine how its components should be initially scheduled on available Grid resources. When the characteristics of a parallel application (e.g. task computational cost, amount of data exchanged among tasks, task dependencies) are known before the application execution, a static scheduling approach can be profitably exploited. Static scheduling usually does not imply overheads on the execution time of the scheduled application, and, therefore, more complex scheduling solutions than the dynamic ones can be adopted. However, in the case of non-dedicated production Grids, the scheduling policy must consider the aging effect on the resources state. High-complexity solutions could lead to wrong scheduling decisions, due to load variations in the system or dynamic changes of the resource availability. On the other hand, re-scheduling requires the ability to modify the application tasks schedule during their execution to cope with dynamic changes of the resource availability to sustain the required performance, or to improve the current application/system performance. Re-scheduling can include changing the machines on which the application processes are executing (migration) and/or changing the scheduling of data to those machines (dynamic load balancing). Dynamic task replication is also a popular technique used to circumvent runtime scheduling problems [7, 11].

To this end performance monitoring and component performance models should be exploited to deal with dynamic changes in the resources capacities and availability. Moreover, the system should exhibit the ability to completely freeze an application, and continue its execution on a different set of resources (checkpointing/migration). Such a technique is useful, not only when the scheduled resources are not able to guarantee anymore the required application performance contract, but also in the case of resource failure.

When performance falls below expectations, the re-scheduling mechanism must determine whether re-scheduling is profitable, and, if so, what new schedule should be used. Since re-scheduling could be a very expensive operation, the launch-time scheduling has to be "optimized" in order to minimize the re-scheduling activity.

In this paper we investigate the issues related to the launch-time scheduling of parallel applications on wide-area Grids. We propose a heuristics, called WASE (Wide Area Scheduling Heuristics), exploiting a TIG (Task Interaction Graphs) [26] as application model and considering large computational Grids made up of not-dedicated computational resources. The goals of such heuristics is threefold: to meet the minimal task computational requirement (soft QoS), to maximize the throughput between communicating tasks, and to evaluate on-the-fly the resource availability to minimize the aging effect on the resources state.

The rest of this paper is organized as follows. Section 2 highlights current efforts in the are of multi-component, parallel applications scheduling. Section 3 gives a description of the problem, Section 4 describe our heuristics, Section 5 outlines and evaluates the proposed heuristics through simulation, and by scheduling a real application on a real Grid. Finally, conclusion and future works are given in Section 6.

2 Related Work

Two important current Grid scheduling efforts in the area of multi-components resource-intensive parallel applications are AppLeS (Application Level Scheduling) [4] and GrADS (Grid Application Development Software) [10] projects. AppLeS proposes a methodology for adaptive application scheduling on heterogeneous computing platforms to meet some application-specific metrics, such as the minimization of the application completion time. The AppLeS approach exploits static and dynamic resource information, performance predictions, application and user-specific information, and scheduling techniques that adapt on-the-fly the application execution. In order to make the integration of a scheduling agent in the application easier, some templates for parameter sweep applications (APST), master/worker applications (AMWAT), and for moldable jobs on space-shared parallel supercomputers (SA) were provided. The goal of GrADS is to realize a Grid system by providing tools, such as problem solving environments, Grid compilers, schedulers, performance monitors, to manage all the stages of application development and execution. Exploiting GrADS, users will only concentrate on high-level application design without putting attention to the peculiarities of the Grid computing platform used. The scheduler is a key component of the GrADS system. In GrADS, scheduling decisions are taken by exploiting application characteristics and requirements in order to obtain the best application execution time. The scheduler is structured according to three distinct phases: launch-time scheduling, re-scheduling and meta-scheduling. At launch-time the application execution is started on the selected Grid resources and a real-time performance monitor is used to track program performance and to detect violation of performance guarantees. Performance guarantees are formalized in a performance contract. In the case of a performance contract violation, the re-scheduler is invoked to evaluate alternative schedules. Meta-scheduling involves the coordination of schedules for multiple applications running on the same Grid at once. It implements scheduling policies for balancing the interests of different applications. Monitoring and Discovery Service (MDS) [9], Network Weather Service (NWS), ClassAds/Matchmaking approach [24], Globus Toolkit [16] and GridFTP for file transfer [3] are the existing software components used to implement the GrADS system.

Moreover, in the past, a number of static scheduling heuristics for heterogeneous computing platforms have been proposed to approximate optimal scheduling solutions. These exploit several techniques such as list scheduling (e.g [20, 27]), clustering (e.g [5, 15], evolutionary methods (e.g [23, 30]) and task duplication (e.g. [2, 8]. The objective of almost the totality of these algorithms is to minimize the overall application completion time, and, in general, such methods consider dedicated Grids with a limited number of nodes. Moreover, the in-depth exploration of the solution space to refine the approximation of the optimal solution may require heuristics execution times that can make these algorithms algorithm not exploitable on non-dedicated wide-area Grids. This is due, in general, to the high variability of the Grid resources status during the time that can make the found scheduling solution inefficient. In practice it is not required to find an initial schedule that better approximates the optimal one, but it is sufficient to find a schedule that guarantees the required application performance.

3 Problem Description

3.1 The Application Model

WASE assumes that a parallel application is composed by multiple tasks in continuous execution, and that the communications between tasks are bidirectional and may occur at any time during the application execution. It is also assumed that the tasks computation and communication costs are known before the application execution. This kind of applications require the co-allocation of parallel tasks before starting the execution.

A weighted Task Interaction Graph (TIG) (Fig. 1), denoted as $G_{App} = (N, V)$, is adopted to model a parallel application. In G_{App} a node $n_i \in N$, with $1 \leq i \leq |N|$, models a task, and an undirected edge $a_{ij} \in V$, with $i \neq j$, models a communication occurring between node n_i and node n_j. Nodes have an associated weight representing the amount of computation required to execute the corresponding task, and edges have an associated weight representing the amount of data exchanged between two nodes. Application costs can be obtained either by static program analysis or by executing the code on a reference system.

In general, the practical evaluation of computation and communication costs is not an easy task. Moreover computational cost, power and load should be represented by means of multi-dimensional quantities, in which each dimension represents one aspect of the computation (floating point operations, memory access, I/O operations). In order to simplify this evaluation we propose to exploit both qualitative and quantitative information on the application. The qualitative one is represented by the *Relative Computational Requirements*, $RCR(n_i)$, and the *Relative Quality of Communications*, $RQoC(a_{ij})$. Both are uniformly defined for nodes and edges, respectively, where e.g. $RCR(n_h) = 3$ and $RCR(n_k) = 1.5$ denotes that n_h has a minimal computational requirement twice n_k. Such values relatively specify the required computational work and communication bandwidth only. We assume that at least one node and one edge exist with $RCR = 1$ and $RQoC = 1$ respectively.

The quantitative information is represented by the value $MCR(G_{App})$, which specifies the *Minimal Computational Requirement* of the application. It is expressed in MFlop/s, as well as the computational power provided by the Grid machines, and it represents the minimal computational power required to efficiently run the tasks with $RCR = 1$.

Given the $MCR(G_{App})$ value, it is possible to determine the $MCR(n_i)$ value for each task of the application:

$$MCR(n_i) = MCR(G_{App}) \cdot RCR(n_i) \tag{1}$$

3.2 Platform Model

A *hierarchical graph* (i.e. dendrogram) denoted as $G_{Grid} = (AN, NL)$ is adopted to model the Grid.

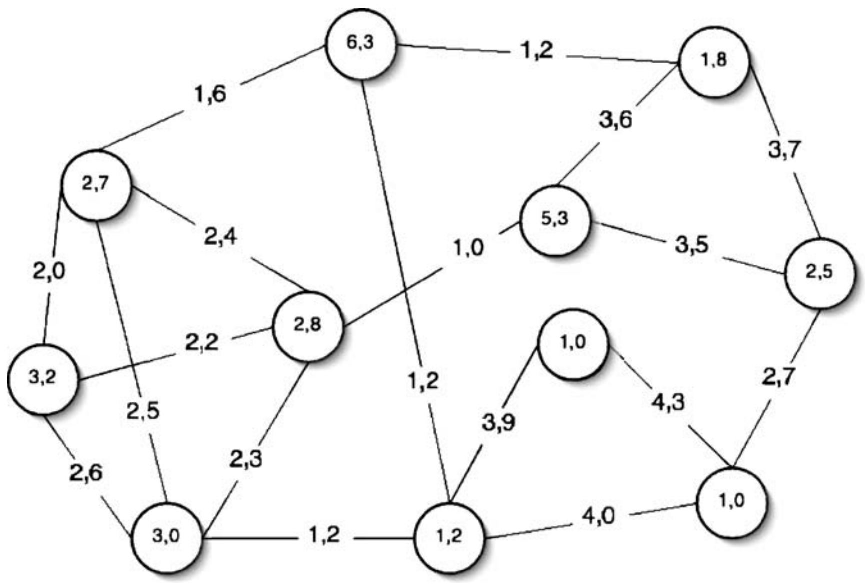


Fig. 1 Example of a weighted task interaction graph

AN is the set of nodes representing sets of computational resources of the Grid and NL abstracts the interconnection networks. We model three different kinds of networks: *Wide Area Networks* (*WAN*), *Metropolitan Area Networks* (*MAN*) and *Local Area Networks* (*LAN*). The root of the graph is a "virtual node" grouping together all the WANs. Every node in the dendrogram represents a set of resources. The LAN set contains the resources to be exploited for tasks execution, while higher-level nodes contain all the resource of the lower-level nodes. Figure 2 shows a dendrogram including two MANs.

A basic assumption of our approach is that intra-LAN communications are characterized by higher bandwidth and a lower level of congestion with respect to any other network communication. It is assumed that for each computational resource $m_{ij} \in an_i \in AN$, its *nominal computational power* P_{ij}^{max} and its *percentage of current computational load* C_{ij}^{load} are known. The *current computational power* of a resource P_{ij}^{cur} is computed as follows:

$$P_{ij}^{cur} = P_{ij}^{max} \cdot \left(1 - C_{ij}^{load}\right) \tag{2}$$

We suppose that each computational resource supports a multiprogramming environment.

The *current* and *average computational power of a LAN an_i* containing $|an_i|$ hosts are computed as follows:

$$P_i^{cur} = \sum_{j=1}^{|an_i|} P_{ij}^{cur} \tag{3}$$

$$\overline{P_i^{cur}} = \frac{P_i^{cur}}{|an_i|} \tag{4}$$

The *current computational availability* of a *MAN* and of a *WAN* can be defined in a similar way.

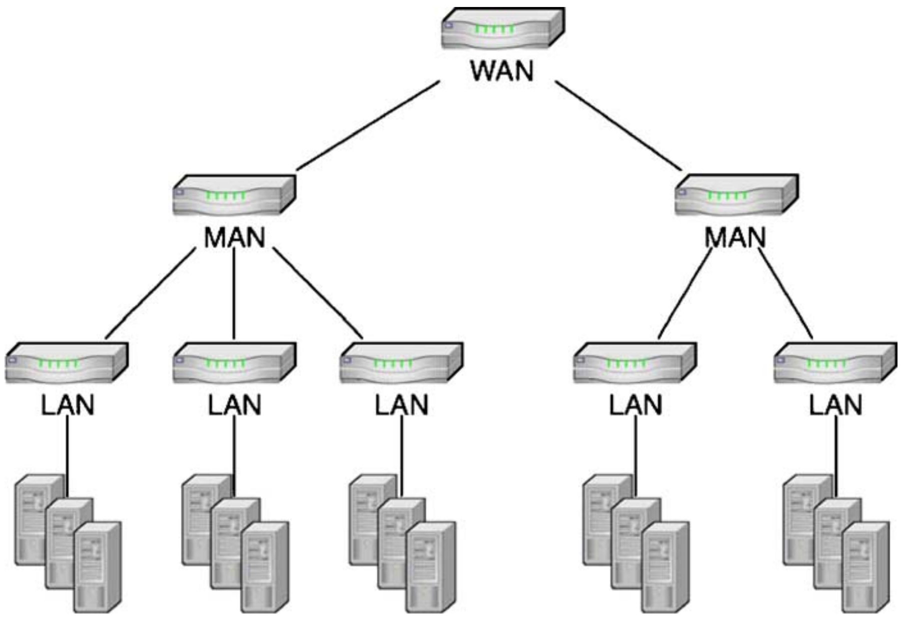
Finally, we also suppose that the *current bandwidth $BW(an_i, an_j)$* between two sibling nodes an_i and an_j is known.

Both the structure of the dendrogram and the characteristics of the resources can be returned by tools like *Network Weather Service* [31] and *TopoMon* [13]. The former returns the network and computational resource performances, while the latter returns an abstraction of the network topology.

4 Algorithm Architecture

The goal of WASE is to provide a *soft QoS* support by fulfilling the application computational requirements and exploiting regions of the Grid

Fig. 2 Grid network topology represented as a dendrogram



network with high communication bandwidths, trying to qualitatively maximize the application communication throughput. The algorithm takes asn input the G_{App} and G_{Grid} graphs, which represent respectively a parallel application and a Grid. The algorithm is structured according to two steps.

Step 1 (Clustering): the goal of this preprocessing step is to elaborate G_{App} to find a set of subgraphs (called clusters) $S = (S_1, \ldots, S_k)$ grouping the tasks with high inter-communication costs. The clustering is carried out by using the ϵ-approximation correlation clustering m proposed in [12]. The goal of the algorithm is to partition the application graph into clusters exploiting a similarity degree associated to each edge (i.e. we try to put into the same subgraph nodes characterized by higher communication cost among them). A positive similarity degree between two nodes means that the algorithm should try to put the two linked nodes in the same cluster, while a negative number will force the two nodes in two different clusters. Basically, the clustering problem is formulated as an integer linear programming problem with boolean variables. This problem is then relaxed and solved with standard LP methods and the solution is rounded to the integer solution exploiting the Region Growing technique [1]. This clustering algorithm does not need any meta-parameter (e.g. number of clusters), and the similarity measure is defined as:

$$w_{ij} = \log\left(\frac{RQoC(a_{ij})}{RQoC^{max} - RQoC(a_{ij})}\right) \tag{5}$$

where $RQoC^{max}$ is the maximum value of $RQoC$ in the application graph. Higher is the communication cost between two tasks (i.e. $RQoC$ value), higher is the confidence that the tasks are similar. In this way, the number of clusters carried out is independent by the Grid size and topology. As a result, we obtain that tasks within the same cluster need higher communication bandwidth than those belonging to different clusters. Therefore, according to the heuristics assumption that the better communication quality characterizes a LAN with respect to MANs and WANs, WASE attempts to allocate all the tasks within a cluster onto the machines of the same LAN.

Step 2 (Scheduling): the tasks within the clusters have to be scheduled onto Grid machines. First, the clusters in S are arranged in descending order with respect to their minimal computational requirement $MCR(S_i)$:

$$MCR(S_i) = MCR(G_{App}) \cdot \sum_{n_k \in S_i} RCR(n_k) \tag{6}$$

Then, the clusters are scheduled onto the machines of suitable LANs to run each cluster's tasks by adopting a *priority-based* policy.

Selecting a cluster and starting from the user LAN (LAN_u), a *Closest First Search* (CFS) is applied to G_{Grid} in order to find a suitable LAN to host all the tasks within the selected cluster. The Closest First Search first visits the local LANs (i.e. the LANs in the user MAN) in descending order of current link bandwidth, measured from the user LAN. Then it moves upward to the user MAN, and again visits the MANs, in the user WAN, in decreasing order of current link bandwidth, measured from the user MAN. Finally, it moves upward visiting the WANs with the usual bandwidth order. Note that, in general, the search does not visit all the nodes of the dendrogram, but it ends when enough resources to satisfy the application requirements are found. With a certain bias, this heuristics tries to minimize the communication overhead and therefore performance degradation by MAN and WAN network links by putting strongly communicating nodes within single LANs.

The LAN_j *suitability* with respect to cluster S_i is computed according to the following formula:

$$\text{Suitability}(S_i, LAN_j) = \frac{\overline{P_j^{cur}[S_i]}}{\overline{MCR(S_i)} + \sigma_{MCR}(S_i)} \tag{7}$$

where $\overline{P_j^{cur}[S_i]}$ is the mean computational power offered by the more powerful $min(|S_i|, |an_j|)$ machines of the LAN,[1] $\overline{MCR(S_i)}$ is the mean minimal computational power requested by the cluster:

$$\overline{MCR(S_i)} = \frac{MCR(S_i)}{|S_i|} \tag{8}$$

[1]Note that if the number of tasks is greater than the number of machines, $\overline{P_j^{cur}[S_i]}$ corresponds to the mean computational of the LAN, as in Eq. 4.

and $\sigma_{MCR}(S_i)$ is the standard deviation of the cluster computational power:

$$\sigma_{MCR}(S_i) = \sqrt{\frac{\sum_{n_k \in S_i} \left(MCR(n_k) - \overline{MCR(S_i)} \right)^2}{|S_i|}} \tag{9}$$

The LAN_j is suitable for S_i if and only if Suitability$(S_i, LAN_j) \geq 1$. The tasks within a cluster are scheduled onto the machines of the first suitable LAN in CFS order.

The tasks are arranged in descending order with respect to their computational cost (i.e. the $RCR(n_i)$ value), and then they are scheduled to the suitable LAN's machines by adopting a *priority-based* policy. The machine onto which schedule a task is selected according to the following parameter:

$$\text{Affinity}(n_i, m_{jk}) = \frac{P_{jk}^{cur}}{MCR(n_i)} \tag{10}$$

The machine m_{jk} is suitable for a task n_i if and only if Affinity$(n_i, m_{jk}) \geq 1$. Every task is scheduled on the machine with the highest affinity rank in the suitable LAN. The allocation of the task n_i on the machine m_{jk} causes a reduction of P_{jk}^{cur} equals to $MCR(n_i)$.

Since statistical information is used to select a suitable LAN, the chosen LAN may not be able to host all the tasks inside a cluster. It could happen that some tasks do not find any suitable machine. To overcome this problem, unallocated tasks are scheduled onto machines of a LAN as close as possible to the other tasks in terms of locality and available bandwidth in the network tree.

This process is repeated until all the cluster's tasks are allocated or until the root of the G_{Grid} graph is reached. In last case the scheduling algorithm fails, ending without a valid allocation.

In Algorithms 1, 2 and 3, the pseudo-code of the heuristics is shown.

Through the identification of the more communicating G_{App}'s regions, and the exploitation of *priority-based* allocation techniques, the algorithm is able to reduce both the *aging* and *stealing* effects on the resources used.

```
 1. Order clusters $S_i \in S$ into $CPQ$
    queue in decreasing
    order of $MCR(S_i)$ values;
 2. while (!$CPQ$.empty()) {
 3.   $S_l$ = $CPQ$.removeFirst();
 4.   Get first $LAN_i \in G_{Grid}$ such
      that
      Suitability$(LAN_i, S_l) \geq 1$ by
      visiting $G_{Grid}$ from $LAN_u$
      in a $CFS$ fashion;
 5.   if ($LAN_i \neq$ null)
 6.     $TBA$ = SIMPLE-TASKMAP
        ($LAN_i, S_l$);
 7.   else
 8.     Exit with ERROR;
 9.   if ($TBA \neq \varnothing$)
10.     PROXY-TASKMAP($LAN_i$, $TBA$);
11. }
12.Exit with SUCCESS;
```

Algorithm 1: LAN-SELECT(S, G_{Sys})

```
 1. Order tasks $n_i \in S_l$ into $TPQ$
queue in decreasing
    order of $MCR(n_i)$ values;
 2. while (!$TPQ$.empty()) {
 3.   $n_i$ = $TPQ$.removeFirst();
 4.   Get resource $m_{jk} \in AN_j$ with
highest value
      of Affinity$(n_i, m_{jk})$;
 5.   if (Affinity$(n_i, m_{jk}) \geq 1$) {
 6.     Allocate task $n_i$ onto
resource $m_{jk}$;
 7.     $P_{jk}^{cur} = P_{jk}^{cur} - MCR(n_i)$;
 8.     if ($P_{jk}^{cur} = 0$)
 9.       $G_{Grid}$.remove($m_{jk}$);
10.   } else
11.     Put task $n_i$ into $TBA$ queue;
12. }
15. return $TBA$;
```

Algorithm 2: SIMPLE-TASKMAP(AN_j, S_l)

5 Performance Evaluation

In this section we present the evaluation of the scheduling solution carried out by WASE. The

```
 1. Order all $AN_h$ brothers of $AN_i$
    (with $h \neq i$) into
    $ANPQ$ queue in decreasing
    order of $BW(AN_i, AN_h)$;
 2. while (!$TBA$.empty() $\wedge$
    !$ANPQ$.empty) {
 3.   $AN_h$ = $ANPQ$.removeFirst();
 4.   $TBA$ = SIMPLE-TASKMAP
      ($AN_h, TBA$);
 5. }
 6. if (!$TBA$.empty())
 7.   PROXY-TASKMAP($AN_i$.father(),
      $TBA$);
 8. if (!$TBA$.empty())
 9.   Exit with ERROR;
10. Exit with SUCCESS;
```

Algorithm 3: PROXY-TASKMAP(AN_i, TBA)

objective is to investigate the effect of the application computational and communication requirements on reaching the WASE goals. The evaluation was conducted by simulations applying WASE to a suite of task graphs and Grid platforms randomly generated. Moreover, a more thorough test was conducted by using a real life rendering application.

5.1 Simulation Environment

To carry out a set of meaningful statistical information to evaluate the proposed heuristics a large number of simulations were conducted. The following results are first steps in analyzing the behavior and quality of the presented approach. As there is no existing information on parameterizing the algorithm, we assume several parameter sets as first examples. Thus, it cannot be concluded that these parameters are optimal nor feasible for other job scenarios. Future work will have to further evaluate the influence of these parameters on other workload and Grid configurations.

The simulation environment used to conduct the tests includes the *Cabri-graphs* [6], *Tiers* [14] and *Network Weather Service* (NWS) [31] tools. *Cabri-graphs* was used to generate application graphs. It generates connected and undirected graphs (hence compatible with a TIG model) having a number of nodes varying from 2 to 256. We generated TIGs with 8, 16, 32, 64 and 128 nodes with a variable number of edges. Such sizes seem reasonable because most part of parallel applications follow the "power-of-two" law and their complexity does not excess the 128 nodes [22]. Both the RCRs for the nodes and the $RQoC$s for the edges were randomly generated from a uniform distribution $U(1, 10)$.

In order to evaluate the effect of the tasks communication cost on the LAN configurations, TIGs were clustered changing the correlation function to obtain 1–3, 4–6 and 7–9 tasks per clusters for each generated TIG. Higher is the value of the correlation degree, higher is the communication similarity value and therefore greater is the number of tasks inside a cluster. Moreover, in each test, each clustered TIG was run by adopting four different values of MCR: 25, 50, 75 and 100.

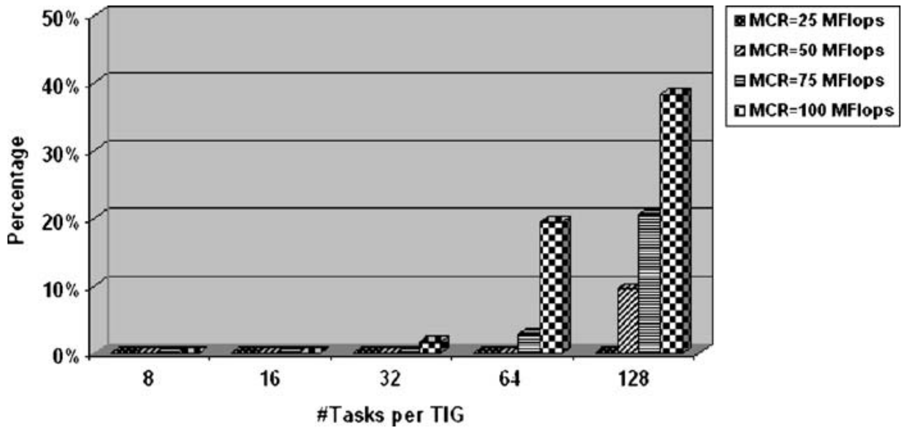


Fig. 3 Percentage of scheduling failures

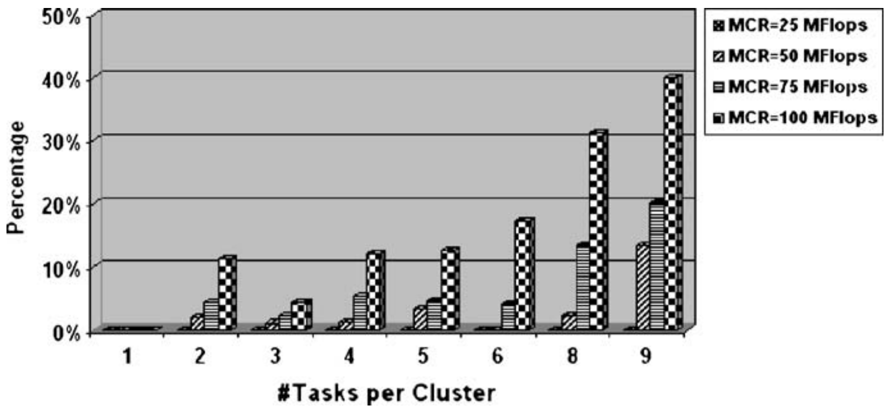


Fig. 4 Percentage of scheduling failures

The Grid platforms have been generated to satisfy the following constraints:

1. Grids structured as real wide area Grid by adopting specific topologies and realistic features for the computational machines and communication links;
2. The Grid topology has to be organized with a hierarchical structure (i.e. WANs, MANs and LANs);
3. The values of the bandwidth and the latency of each link must reflect the ones of real Grid environments, especially by considering the different communication bandwidth among LAN, MAN and WAN networks.

The previous requirements are satisfied by the *Tiers* tool. Tiers allows the generation of a large number of Grid systems with different topologies having a number of hosts varying from 16 to 1,024.

The loads of machines and links have been obtained by referring to the traces stored in the *Network Weather Service* (NWS) [31] web site, where information of several hosts connected in different networks are available.

To run all the tests, 240 application TIGs and 15 Grids were generated. All the 240 TIGs were generated with a random number of nodes (varying from 8 to 128) and a random number of edges. The values representing the task computation and edge communication requirements have been randomly generated as well. We generated Grids with 32, 64, 128, 256 and 512 machines, grouped in LANs with 4, 8 and 16 machines. Then, a value representing the computational power has been associated to each host on the basis of

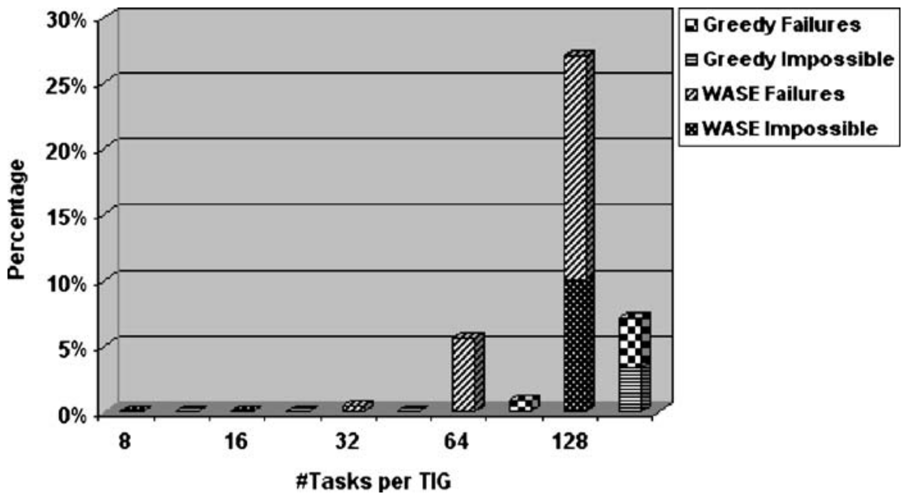


Fig. 5 Comparison of percentage of scheduling failures

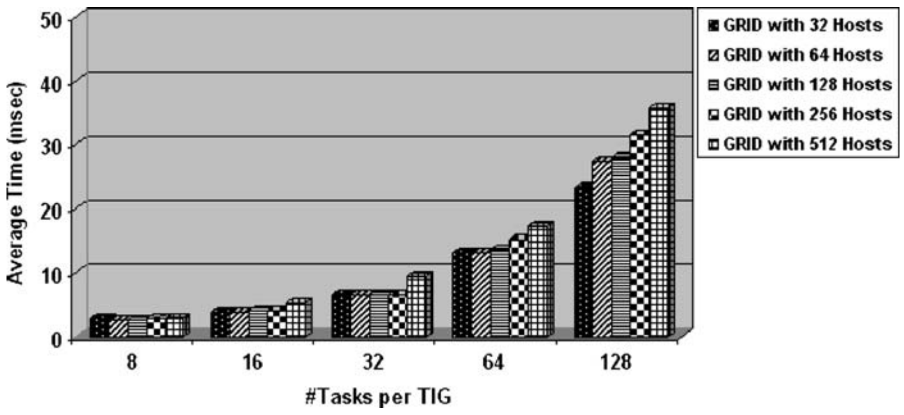


Fig. 6 Average scheduling time

a uniform distribution with values from 600 to 1,300 MFlop/s. In all the tests, about 3,600 trials were executed, and the average values for each test (about 720 trials) are shown.

5.2 Performance Metrics

To validate WASE we implemented a greedy *best fit* scheduling and we applied such algorithm to every test case. To evaluate the schedules carried out by WASE we exploited different criteria: the percentage of mapping failures, the task-machine affinity, the LAN hit ratio and the intra-cluster "distance" among tasks allocated on different an_i.

The percentage of scheduling failures indicates the ratio of test applications that both algorithms failed to schedule. This can be due to the fact that-it was impossible to find a schedule for the application or the selected algorithm failed to find one of the existing solutions. In the following criteria, the average values are computed discarding the simulations resulting in scheduling failures.

The task-machine affinity represents how many times the power of the machine on which a task is allocated is greater than the task minimal computation requirement. Hence, this parameter gives us an accurate estimation of the satisfaction level obtained by WASE in allocating a task.

The LAN hit ratio gives us how many tasks of a cluster are allocated onto the machines of the suitable LAN. According to the WASE hypothesis, the more the LAN hit ratio increases the more the throughput between communicating tasks increases.

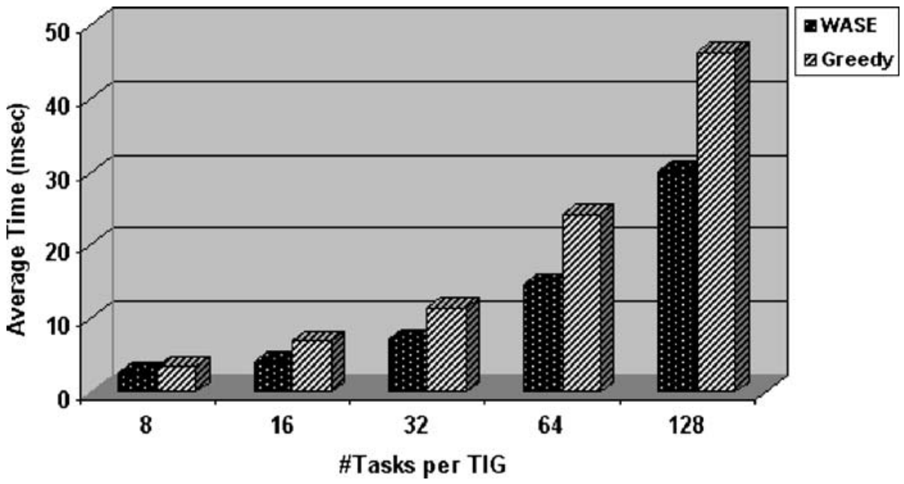


Fig. 7 Comparison of average scheduling times

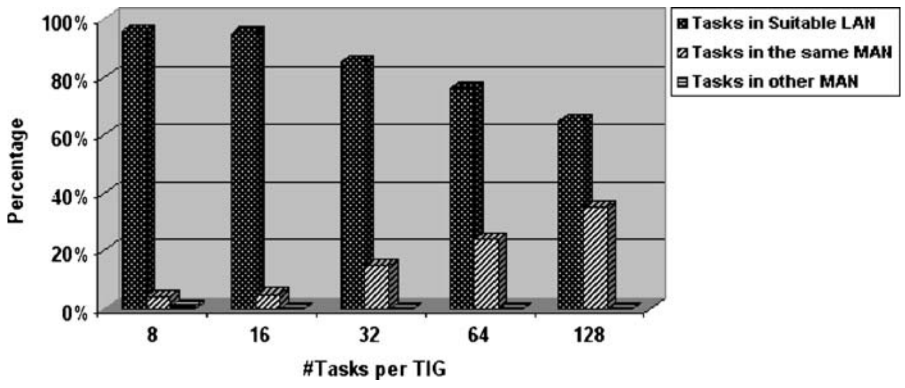


Fig. 8 Scheduling of TIG (four hosts per LAN)

Finally, the intra-cluster "distance" represents a scheduling distribution to estimate the reduction of the communication quality among the tasks of a cluster when some of them are not allocated in the suitable LAN.

These results have been obtained running the algorithm on a Pentium4 processor with a clock frequency of 3.0 GHz and a memory of 1 GB.

5.3 Evaluation for Synthetic Grid Scenarios

To evaluate the schedules carried out by WASE we first evaluate the percentage of scheduling failures. Figures 3 and 4 show the percentage of scheduling failures with respect to the number of tasks per TIG and to the number of tasks per cluster, respectively. As expected, an exponential increase of the number of tasks or the size of a cluster causes an exponential increment of the number of failures. In Fig. 5 WASE is compared with the scheduling carried out by a greedy scheduling algorithm. This algorithm simply orders the task queue in decreasing order of RCR and the machine queue in decreasing order of computational power and it performs a *best fit* scheduling. As can be seen, in the simulations, the percentage of failures of the greedy algorithms is lower than the percentage failure of WASE.

In the following evaluations, the average values are computed discarding the simulations resulting in scheduling failures.

In Fig. 6 the average scheduling times of WASE that was needed for conducting the simulation on the aforementioned machine are shown. It grows exponentially with the number of tasks per TIG. In Fig. 7 we compare the average scheduling time of WASE with those obtained by a greedy algorithm on the same simulations. It can be seen

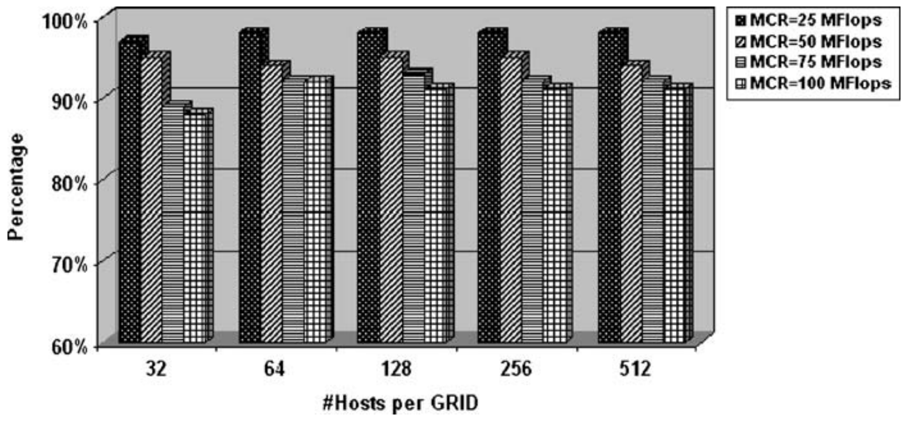


Fig. 9 Average LAN Hit Ratio (WASE)

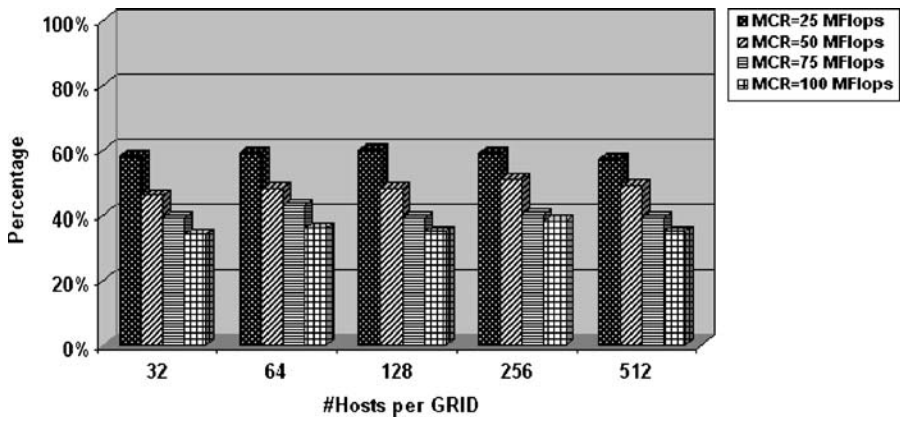


Fig. 10 Average LAN Hit Ratio (Greedy)

that WASE presents better values than the greedy heuristics. It is mainly due to the fact that the greedy one must find the "best fit" on the queue of the machines for each task, while WASE just tries to exploit user LAN resources.

This is shown in Fig. 8. For realistic Grid systems as the ones generated by Tiers, WASE was always able to schedule tasks in the first suitable LAN or in LANs belonging to the same MAN, efficiently exploiting the Grid communication resources.

The LAN hit ratio is the percentage of the tasks of a cluster that are allocated into the suitable LAN. Since the allocation of the tasks in the same LAN allows to obtain a better communication quality, higher values of the LAN hit ratio mean an improvement of the application throughput.

Figures 9 and 10 show the results of the average LAN hit ratio using WASE and greedy algorithm, respectively. In such tests we can observe that the LAN hit ratio of WASE is around 90%, while the one of the greedy algorithm spans from 20 to 60%. This result demonstrate that WASE is able to allocate the tasks of a cluster in the same LAN and this allows to avoid a degradation of the communication quality.

Finally, Fig. 11 shows a limit of the proposed solution: changing the size of the Grid, the minimum task-machine affinity does not change. This is due to the fact that the algorithm always search for good resources starting from the user LAN and moving in nearby LANs/MANs. This behavior prevents the algorithm to explore distant LANs, even if such LANs are able to host a whole cluster

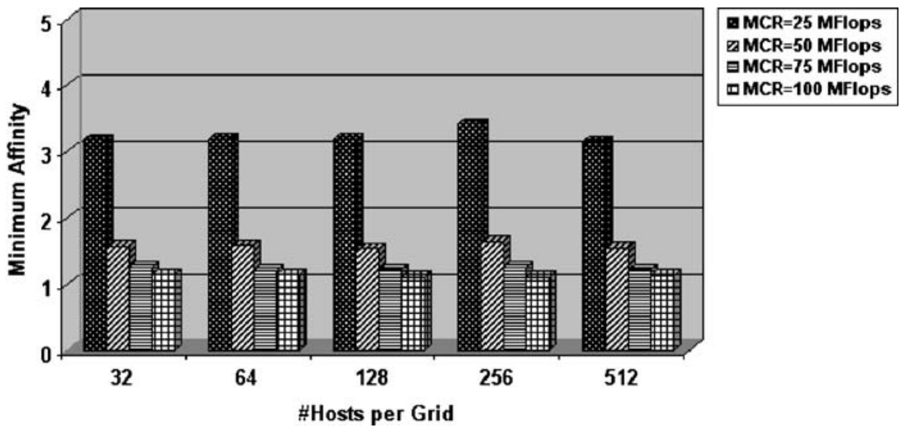


Fig. 11 Minimum Task Machine Affinity (TIGs with 8, 16, 32, 64 and 128 tasks)

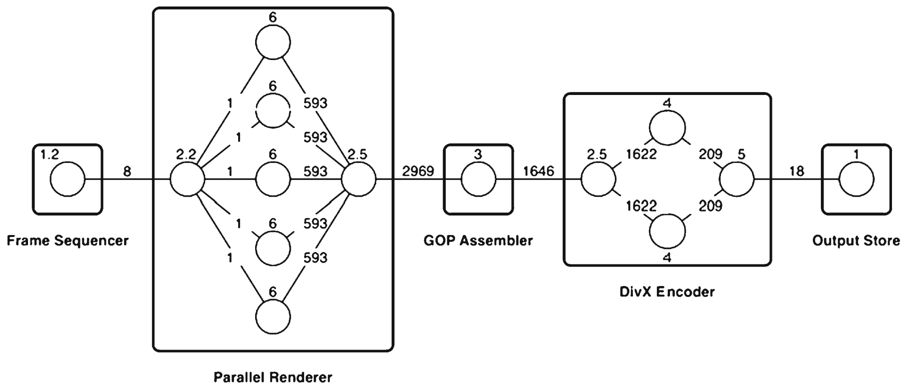


Fig. 12 Graph of the render-encode application

of tasks. Future improvements of our solution are under investigation to find a better compromise between the average scheduling time and the minimum task-machine affinity.

5.4 Evaluation for a Parallel Rendering Application

Figure 12 shows the structure and the RCR and $RQoC$ values of the real application, a rendering one, structured according to the pipeline program paradigm, used in the last test. The first sequential stage requests the rendering of a sequence of scenes. The second one is a data-parallel module composed by an emitter, a collector and five workers that render each scene (exploiting the PovRay rendering engine) interpreting a script describing the 3D model of objects, their positions and motion. The third stage collects images rendered by the second one, and builds Groups Of Pictures(GOP) that are sent to the fourth stage performing DivX compression in a data-parallel module composed by two workers. The last stage collects DivX compressed pieces and stores them in an AVI output file. RCR and $RQoC$ values have been collected through a short profiling execution of the application on a reference architecture.

In Fig. 13 the testbed exploited for the algorithm evaluation is shown. This is a Grid composed by 23 heterogeneous computational resources (seven workstations and two clusters) distributed in three LANs connected through 1 Gb/s optical links. The nominal computational power of each machines is listed in Table 1. The average initial load on each one has been set to 30%.

We have applied the clustering algorithm with three different similarity thresholds to obtain

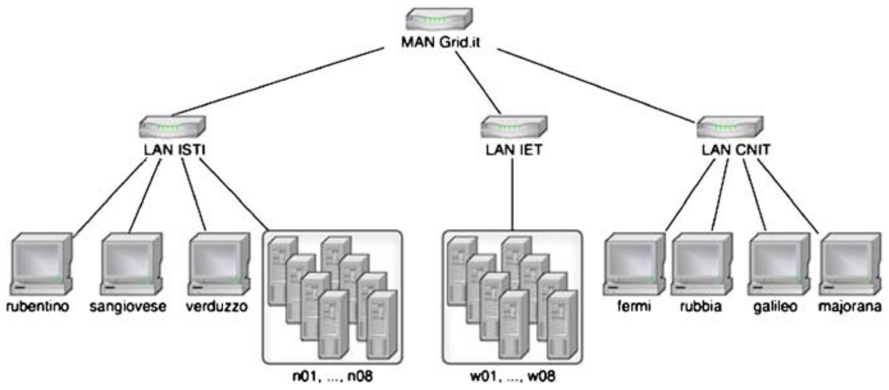


Fig. 13 Graph of real Grid testbed

Table 1 Computational power of resources in the Grid testbed

Machine	P_i (Mflop/s)	Machine	P_i (Mflop/s)
rubentino	1,000	w01	650
sangiovese	950	w02	625
verduzzo	1,000	w03	700
n01	1,350	w04	650
n02	950	w05	1,250
n03	950	w06	1,300
n04	950	w07	1,100
n05	950	w08	700
n06	950	fermi	1,350
n07	950	galileo	1,350
n08	950	rubbia	1,300
		segre	1,325

different clusters as shown in Fig. 14. Then, WASE has been applied with three different MCR values. In any case, we have considered ISTI as the user LAN. The allocation results, obtained as function of the number of clusters, are shown in Table 2.

We can observe that an increase of the MCR value causes a decrease in the average task-machine affinity. This is due to the increase of the computational power requested by the tasks within a cluster. Moreover, the more the number of clusters increase, the more the inter-LAN com-break munications increases (more clusters, more edges on the cuts). This is a kind of optimality parameter for the results of the allocation phase. In fact, let us consider the clustering result as the best one for the application. If the scheduling phase is able to schedule every single cluster completely

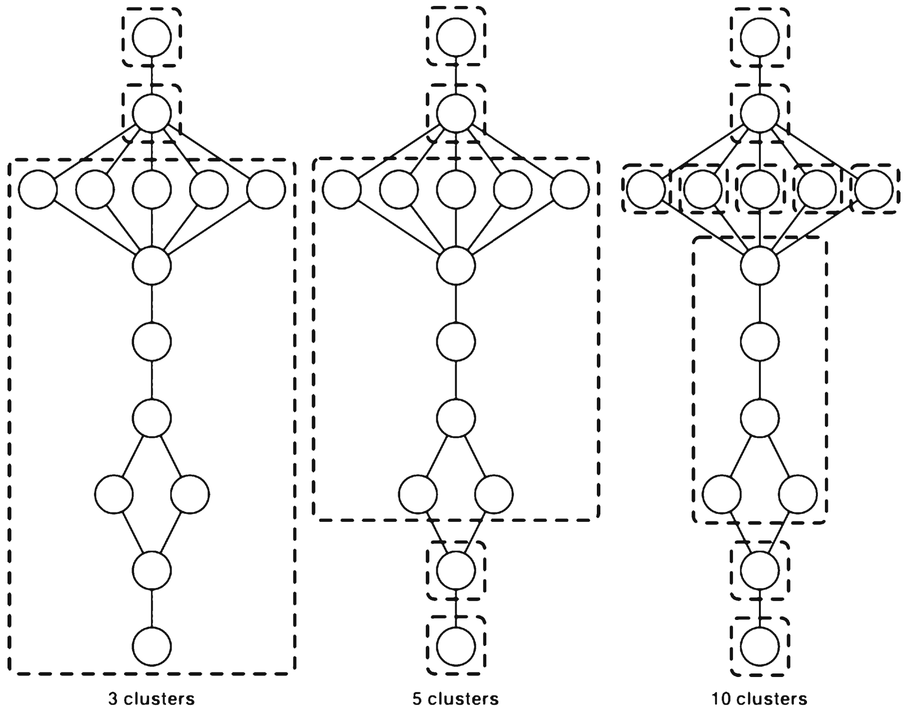


Fig. 14 Different clustering results of the test application

Table 2 Allocation results for the use case application

	MCR	LAN ISTI	LAN CNIT	LAN IET	Average affinity
3 clusters	50	14	0	0	13.35
	100	14	0	0	5.63
	150	8	1	5	4.32
5 clusters	50	14	0	0	13.93
	100	14	0	0	6.72
	150	6	1	7	4.48
10 clusters	50	14	0	0	14.75
	100	14	0	0	7.00
	150	9	1	4	4.82

	Inter LAN communication (clustering)(%)	Inter LAN communication (scheduling)(%)
3 clusters	0.12	0.00
		0.00
		20.81
5 clusters	3.98	0.00
		0.00
		24.48
10 clusters	30.27	0.00
		0.00
		26.04

onto a single LAN, eventually we cannot obtain an inter-LAN communication value worse than the previous one. During the scheduling phases, the inter-LAN communication values are influenced by two effects:

1. A single cluster may be too "large" to be scheduled on a single LAN, and it must be split across several LANs. It happens with few and big clusters, and this effect worsen the final inter-LAN communication value;
2. Several clusters may be scheduled on the same LAN, if there are available resources, improving the final inter-LAN communication value.

We can see both effects in the test results. In the case of three clusters, the scheduling algorithm is forced to split the big cluster in several LANs, while in the case of 10 clusters, the heuristics is able to completely allocate the largest cluster on a LAN and to allocate several small clusters on the same LANs.

6 Conclusions and Future Work

In this paper a new heuristics to schedule parallel applications on Grid platforms is proposed. The heuristics is oriented to large computational Grids made up of dynamic not-dedicated resources that are generally known as *wide-area Grids*. The allocation of a parallel application is conducted to satisfy the minimal computational requirements (soft QoS) of its tasks, and by exploiting the intra-LAN communications in order to maximize the throughput between communicating tasks. A further heuristics goal is to quickly allocate a task by evaluating on-the-fly the resource availability to minimize the aging effect on the resources state. The heuristics exploits a technique based on three elements: task computation and communication costs, clustering of the application graph to find regions with high communication costs, and a dendrogram representing a Grid where the resources are hierarchically grouped in LANs, MANs and WANs. The allocation strategy is carried out by exploiting a priority-based technique to satisfy the task minimum computational power requirement. Furthermore, in order to maximize the throughput between communicating tasks the heuristics tries to allocate tasks of a given cluster by exploiting the machines within a LAN, where the better communication quality is assumed.

In order to evaluate the quality of the scheduling algorithm, several tests were conducted by simulating the execution of a large number of parallel applications on a number of Grids both randomly generated. Moreover, a further test was conducted by using a real application on a real Grid.

The performance analysis highlights that the scheduling quality is satisfactory even when the number of the application tasks and the number of hosts increase.

The presented evaluation examples indicate that the WASE algorithm is very suitable for Grid use. However, in future work more thorough simulations and evaluations should be conducted to further analyze the behavior and parametrization for more Grid structures and application examples.

Moreover, the algorithm can be improved in several directions. A first development should be to refine the suitability function to a better choose of the LAN suitable for a cluster. This improvement will reduce the allocation of the tasks belonging to a same cluster on different LAN. Another development should be the addition of a soft QoS based on minimum requirement of the link bandwidth in order to minimize the communications times requested for the data transfers between two interacting tasks.

Finally, the last improvement should be the support of the dynamic spawning of new tasks of a running application. In fact, a large number of parallel applications are able, at running time, to carry out a dynamic load-balancing by creating new components or by splitting a big component in several sub-components. With this support the functionalities of the heuristics could be extended to new classes of parallel applications.

Acknowledgements This work has been supported by: the Italian MIUR FIRB Grid.it project, No. RBNE-01KNFP, on High-performance Grid platforms and tools, and the European CoreGRID NoE (European Research Network on Foundations, Software Infrastructures and Applications for Large Scale, Distributed, GRID and Peer-to-Peer Technologies, contract no. IST-2002-004265).

References

1. Adams, R., Bischof, L.: Seeded region growing. IEEE Trans. Pattern Anal. Mach. Intell. **16**(6), 641–647 (1994)
2. Ahmad, I., Kwok, Y.-K.: On exploiting task duplication in parallel program scheduling. IEEE Trans. Parallel Distrib. Syst. **9**(9), 872–892 (1998)
3. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Data management and transfer in high-performance computational Grid environments. Parallel Comput. **28**(5), 749–771 (2002)
4. Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A., Zagorodnov, D.: Adaptive computing on the Grid using apples. IEEE Trans. Parallel Distrib. Syst. **14**(4), 369–382 (2003)
5. Bowen, N.S., Nikolaou, C.N., Ghafoor, A.: On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. IEEE Trans. Comput. **41**(3), 257–273 (1992)
6. Cabri-graphs: IMAG. http://www-cabri.imag.fr/CabriGraphes/ (1998)
7. Carter, B.R., Watson, D.W., Freund, R.F., E. K., Mirabile, F., Siegel, H.J.: Generational scheduling for dynamic task management in heterogeneous computing systems. Inf. Sci. **106**(3,4), 219–236 (1998)
8. Choe, T.-Y., Park, C.-I.: A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. In: Proceedings of the 14th International Parallel and Distributed Processing Symposium. Washington, DC, USA (2000)
9. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid information services for distributed resource sharing. In: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing. Washington, DC, USA (2001)
10. Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., Foster, I.: Scheduling in the Grid application development software project. In: Resource Management in the Grid. Hingham, MA, USA (2003)
11. de O. Lucchese, F., Yero, E.J.H., Sambatti, F.S., Henriques, M.A.A.: An adaptive scheduler for Grids. Journal of Grid Computing **V4**(1), 1–17 (2006)
12. Demaine, E., Immorlica, N.: Correlation clustering with partial information. In: Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems. Berlin, Germany (2003)
13. den Burger, M., Kielmann, T., Bal, H.: TOPOMON: A monitoring tool for Grid network topology. In:

Proceedings of the International Conference on Computational Science. Berlin, Germany (2002)
14. Doar, M.: A better model for generating test networks (1996)
15. Eshaghian, M.M.: Heterogeneous computing. Artech House, Norwood, MA, USA (1996)
16. Foster, I., Kesselman, C.: The globus project: a status report. Future Gener. Comput. Syst. **15**(5,6), 607–621 (1999a)
17. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA, USA (1999b)
18. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. **15**(3), 200–222 (2001)
19. INRIA: ProActive. http://www-sop.inria.fr/oasis/ProActive (1999)
20. Iverson, M., Ozguner, F., Follen, G.: Parallelizing existing applications in a distributed heterogeneous environment. In: Proceedings of the 4th Heterogeneous Computing Workshop (1995)
21. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of Grid resource management systems for distributed computing. Software Practice and Experience **32**(2), 135–164 (2002)
22. Li, H., Groep, D., Wolters, L.: Workload characteristics of a multi-cluster supercomputer. In: Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel processing. Berlin, Germany (2004)
23. Lo, V.M.: Heuristic algorithms for task assignment in Distributed Systems. IEEE Trans. Comput. **37**(11), 1384–1397 (1988)
24. Raman, R., Livny, M., Solomon, M.: Matchmaking: an extensible framework for distributed resource management. Cluster Comput. **2**(2), 129–138 (1999)
25. Schopf, J.M.: General Architecture for Scheduling on the Grid. Technical report, Argonne National Laboratory (2002)
26. Sinnen, O., Sousa, L.: A classification of graph theoretic models for parallel computing. Technical report, Instituto Superior Tecnico, Technical University of Lisbon, Portugal (1999)
27. Topcuouglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
28. Vadhiyar, S., Dongarra, J.: A metascheduler for the Grid. In: Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing. Washington, DC, USA (2002)
29. Vanneschi, M.: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Comput. **28**(12), 1709–1732 (2002)
30. Wang, L., Siegel, H.J., Roychowdhury, V.R., Maciejewski, A.A.: Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. J. Parallel Distrib. Comput. **47**(1), 8–22 (1997)
31. Wolski, R., Spring, N.T., Hayes, J.: The network weather service: a distributed resource performance forecasting service for metacomputing. Future Gener. Comput. Syst. **15**(5,6), 757–768 (1999)