Future Generation Computer Systems 37 (2014) 1–13

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Two new fast heuristics for mapping parallel applications on cloud computing

I. De Falco, U. Scafuri, E. Tarantino *

Institute of High Performance Computing and Networking, National Research Council of Italy (ICAR-CNR), Via P. Castellino 111, 80131 Naples, Italy

HIGHLIGHTS

- The paper deal with the mapping problem.
- Specific reference is made to task interaction graph applications.
- Two new fast heuristics are proposed.
- These heuristics are improvements of the classical Min–min and Max–min algorithms.
- The results demonstrate the effectiveness of the proposed algorithms.

ARTICLE INFO

Article history:
Received 19 December 2012
Received in revised form
16 December 2013
Accepted 24 February 2014
Available online 6 March 2014

Keywords:
Cloud computing
Mapping
Communicating tasks
Heuristics

ABSTRACT

In this paper two new heuristics, named Min–min-C and Max–min-C, are proposed able to provide near-optimal solutions to the mapping of parallel applications, modeled as Task Interaction Graphs, on computational clouds. The aim of these heuristics is to determine mapping solutions which allow exploiting at best the available cloud resources to execute such applications concurrently with the other cloud services.

Differently from their originating Min–min and Max–min models, the two introduced heuristics take also communications into account. Their effectiveness is assessed on a set of artificial mapping problems differing in applications and in node working conditions. The analysis, carried out also by means of statistical tests, reveals the robustness of the two algorithms proposed in coping with the mapping of small- and medium-sized high performance computing applications on non-dedicated cloud nodes.

1. Introduction

The cloud paradigm [1–3], commercially supported by important firms as for instance Google [4], Amazon [5], and Microsoft [6], refers to the use of computing resources (hardware and software) delivered as a service over a network.

The current cloud systems offer on-demand a broad range of virtualized services which can be classified into three major models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). IaaS refers to the practice of delivering on demand IT infrastructure as a commodity to customers. PaaS provides a development platform in which customers can create and execute their own applications. SaaS endows the user with an integrated service comprising hardware, development platforms, and applications. All these models allow users to have at their disposal the resources needed without having any knowledge about their numbers, characteristics, and location.

Although unsuitable to efficiently solve compute-intensive parallel applications, many efforts are being dedicated by manufacturers and scientists to provision the cloud systems with new functionalities. These capabilities allow executing in reasonable times small- and medium-sized high performance computing (HPC) applications [7–9].

For the contemporary cloud systems the customers who have to execute in parallel a multitask application can already negotiate, by an IaaS contract, the leasing of virtual machines for a prefixed time interval. Unfortunately such a leasing generally provides functionalities relatively to the networking, the data storage, the physical servers, and the virtualization software. The management of the operating system, of the middleware and of the runtime phase, the choice of the number of virtual machines with specific HW/SW characteristics, the mapping of the tasks on these virtual machines, and the scheduling are left to the customers.

An alternative is that the customers want to commit to a cloud middleware the management and the execution of their multitask

* Corresponding author. Tel.: +39 081 6139525; fax: +39 081 6139531.
E-mail address: ernesto.tarantino@na.icar.cnr.it (E. Tarantino).

http://dx.doi.org/10.1016/j.future.2014.02.019
0167-739X/

applications concurrently with the current workload of the cloud system. This service can be negotiated by a new form of PaaS contract. To support this new contract the cloud middleware, in addition to mapping on the physical resources the virtual machines and all the other services requested by other customers, must also establish the optimal task/node deployment for the submitted multitask applications.

Unfortunately, most of the mapping tools described in scientific literature make reference to the allocation of independent tasks. Only a few of them can deal with applications with communicating tasks [10–13], yet they all can work only as long as applications are modeled as Direct Acyclic Graphs (DAGs) [14–16]. This kind of applications is much less general and flexible than multitask applications modeled as Task Interaction Graphs (TIGs) [17,18]. In the TIG model all the tasks are considered simultaneously executable and the communications can take place at any time, in general according to iterative and non-deterministic patterns. This means that there is no precedence relationship among tasks: each task cooperates with its neighbors [19].

In this paper we wish to move a first step towards filling this gap by introducing two mapping heuristics to efficiently allocate TIG applications on non-dedicated cloud resources. As shown in [20,21], given the NP-complete nature of the mapping, metaheuristic algorithms are the most appropriate to attain approximate solutions that meet the application requirements in a reasonable time [22–24].

In [25] a wide comparison among eleven heuristics is reported for allocation of independent tasks. The conclusions state that for the different situations, implementations, and parameter values used there, Genetic Algorithm (GA) consistently gave the best results. The average performance of the relatively simple Min–min [26] heuristic was always within 12% of the GA heuristic. Also Max–min [27] turns out to be quite effective and, as Min–min, has the advantages of not necessitating parameter tuning and of being much faster in terms of convergence times. Moreover, more recently, Luo et al. [28] have taken 20 different fast greedy heuristics into account when aiming to allocate independent tasks. Their results confirm that the classical Min–min performs very well, since it is within the three best algorithms.

Since literature assesses the robustness and the effectiveness of Min–min and Max–min for independent tasks, we have decided to improve them so that they can account for communication times too. It should be emphasized that these seminal algorithms take care of computations only, so communications among tasks are neglected whenever a node has to be chosen as the most suitable allocation for a given task. This is due to the sequential nature of these algorithms, which iteratively place one task at a time, so communication times cannot be considered if all the tasks have not yet been mapped.

To overcome this drawback a brand-new way of taking communications into account in algorithms such as Min–min and Max–min is presented. Namely, two new heuristics, Min–min-C and Max–min-C, are introduced. They are based on the classical Min–min and Max–min algorithms yet communications too are considered, and seem therefore very promising for an effective mapping of communicating tasks making up parallel applications modeled as TIGs.

The effectiveness of our two algorithms is evaluated by performing the mapping of artificial applications on a cloud infrastructure at different workload operating conditions, and is assessed against that of the two original algorithms.

Paper structure is as follows: Section 2 reports on the related research; Section 3 presents the working environment, while Section 4 explains our algorithms. In Section 5 the test problems experienced are reported, the results attained are discussed, and a statistical analysis is presented. Finally in Section 6 conclusions are given, and a discussion on the approach proposed and on the open problems to deal with is outlined.

2. Related research

The selection of the cloud resources that, on the basis of physical characteristics (computational power, frequency, memory, bandwidth, …) and load (known or estimated), better support the services as they are negotiated by the customers is nearly always a problem of considerable difficulty.

In [29,30] the mapping is performed manually through a laborious procedure whose results are particularly error-prone when the number of virtual machines reaches hundreds or thousands of nodes.

Fang et al. in [23] discuss a mapping mechanism related to independent tasks, based on the two levels of load balance, which considers the flexibility and virtualization in cloud computing. The first-level scheduling is from the users' application to the virtual machine, and creates the description of a virtual machine according to the resources and other configuration information demanded by the application tasks. The second level is from the virtual machine to host resources, and finds appropriate resources for the virtual machine in the host resources under certain rules, based on the description of the virtual machine for each task. The load of the virtual machine is evaluated by the predicted execution time of the tasks running on it.

In [31,32] approaches for a rule-based mapping which are able to automatically adapt the mapping between virtual machines and physical hosts' resources are advanced. The authors extended the open source solution Eucalyptus and the papers differ for the performance evaluation metrics of the chosen mapping policies: maximizing computation performance and virtual machine locality to achieve a high performance, and minimizing energy consumption in the first case, while the second approach includes the waiting time, the turnaround time, and the response time of the proposed algorithm.

Unfortunately the criteria and the mapping algorithms presented up to now cannot be exploited to efficiently execute TIG applications on cloud nodes. In fact, the mapping of this kind of applications introduces further degrees of complexity if the cloud resources, besides being heterogeneous and geographically dispersed, have features that vary even substantially over time as the local loads and the network bandwidth dynamically change [26,33]. The same holds for the classical mapping algorithms which, known as NP-complete already on traditional parallel and distributed systems, cannot work adequately in heterogeneous environments [34], such as clouds are.

Considered the NP-nature of the mapping problem, a metaheuristic algorithm has been investigated by Mehdi et al. [24]. To speed up the mapping process and ensure the fulfillment of all task deadlines and QoS requirements, the authors introduce a fast algorithm that can find a mapping using genetic algorithms with 'exist if satisfy' condition. Mapping time and makespan are the performance metrics that are used to evaluate the proposed system.

A very recent and interesting paper is [13], in which two greedy algorithms are used to generate the static allocation for the tasks composing a DAG application in a cloud. One is called Cloud List Scheduling (CLS), and the other Cloud Min–Min Scheduler (CMMS). Since the original Min–min algorithm does not consider the dependences among tasks, in CMMS the mappable task set must be updated in every scheduling step to maintain the task dependences. Namely, tasks in the mappable task set are the tasks whose predecessor tasks are all assigned. It is worth noting that this is a first step to extend the use of Min–min to interacting tasks, yet this modification affects DAG structures only, and cannot work for the here considered TIGs, which are a much more general structure.

The problem of allocating TIGs to computing systems is addressed in several papers but limited either to local area networks [35] or to clusters of PCs [36], or to grid environments relatively to dedicated nodes with advance reservation [37], and restricted to the task grouping [38].

3. Our working environment

3.1. HPC parallel applications vs cloud computing

In [39] the cloud computing stack is described as a four-layered architecture of IT infrastructures. At the lowest layer there are the physical resources which, grouped in sites even geographically dispersed, typically communicate each other by channels with a different and frequently low bandwidth. Generally these resources are quite different according to the cloud nature. For example, typical commercial cloud deployments are more likely composed by datacenters with hundreds or thousands of computing nodes that host virtual machines to execute customers' tasks.

To guarantee appropriate services to the customers, the cloud resources are managed by a core middleware layer which, endowed with a wide set of services, provides a suitable runtime environment to adequately support the requests negotiated by the different forms of IaaS, PaaS, and SaaS contracts.

Since, due to the heterogeneity of the resources and the low interconnection bandwidth, the clouds are judged unsuitable for the execution of HPC intensive parallel applications, it is highly probable that these systems will constitute in the future valid, if not the only, platforms able to run small or medium HPC parallel applications.

To execute onto cloud systems this kind of applications three successive phases must be undertaken: submission, mapping or task-to-node allocation, and task scheduling. The first phase presumes that the user provides the code of the application tasks, each of which may have various requirements [40,41]. The second phase, in accordance with the amount, type, and status of the available resources (i.e., taking into account their estimated dynamical workload), requires the cloud manager to select the mapping solutions that match at best the application needs with the available cloud resources. Finally, as the scheduling phase is concerned, it is to note that the execution of MPI communicating tasks on different nodes proceeds only if they are simultaneously scheduled [42–44]. Then, hypothesizing that in cloud systems the features and average loads of all the different nodes are known to the cloud manager, the proposed mapper takes into account such a problem by guaranteeing the co-scheduling. This is effected by selecting only the cloud nodes which, according to their computing workload, have at their disposal the computing power needed to make the tasks assigned to them immediately running.

3.2. Our parallel application mapper

By dropping the other modules of the cloud middleware which support the various IaaS, PaaS, and SaaS services, in Fig. 1 only those involved in the process that implements the service relative to the execution on the cloud of the TIG applications are reported. As it can be noted from the figure, this service requires the presence of two further specific modules: the *Parallel Application Mapper* (*PAM*) of the *Cloud Manager* (*CM*), and the *Parallel Application Queue* (*PAQ*) of the cloud node. In fact the other outlined modules are in any case involved in other cloud services.

The *PAM* interacts with the *Accounting* module to retrieve the information needed for the application to be run, while it obtains the workload of each cloud node by interacting with the *Cloud Resource Status* (*CRS*) tool.

A customer who has to run a TIG application, provides the *Accounting* tool of the *CM* with the task codes and with all the information requested by the pertaining submission form, such as the number of tasks and, for each task, the amount of computations to be executed, the communicating partners and the amount of communications which must be exchanged with each partner. The *Accounting* tool provides the task requirements to the *PAM* that activates its mapping procedure which, on the basis of available

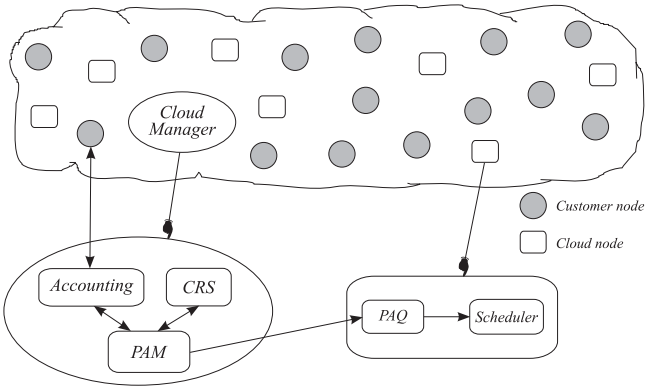

Fig. 1. The TIG application service.

cloud resources provided by the *CRS*, returns to the *Accounting* the allocation retrieved. The expected resource use time and the pertaining cost of this allocation are shown to the customer that can decide whether or not to accept the relative contract.

In case of acceptance, the *Accounting* informs the *PAM* which starts the procedure to submit the task codes to the *PAQ* of each cloud node involved in the agreed mapping solution. As final step, the *PAM* informs the *CRS* of the new load assigned to each of the selected nodes.

The mapping strategy takes into account that the time spent to execute a task on a node is affected by potential blocked states due to the communication timing of the application tasks. If a single task were assigned to a node with suitable available resources, it might be idle waiting for events, e.g., communications, and node resources would remain temporarily unexploited. Then, to exploit at best the available resources, the possibility to allocate on the same node more tasks, belonging to the same or different applications, is allowed so that, during the block-state intervals, the node can be harnessed to concurrently execute another task. When an additional task is allocated on a node, the execution of the new incoming task is evaluated by taking into account the time requested for the execution of the tasks already mapped on the same node.

Since the tasks in *PAQ* can belong to different applications, to guarantee to the user that the execution of his application is carried out on the basis of the submission order, a pre-emptive FIFO scheduling policy is adopted. According to this hypothesis, a process relative to a task of an application in a *PAQ* is *running* if and only if there are no processes which precede it in the queue, while it is descheduled and returns *ready* as soon as a process previously submitted becomes executable. In other words, if at the instant t a process is executing on the node j, such a process releases the use of the processor only for the time intervals in which it results *blocked* either waiting for a communication or in presence of another process which precedes it in the queue. When such a blocked process returns *ready* it becomes immediately *running* if processes with a higher priority in the queue do not exist.

This strategy can fulfill the needs of the cloud manager that can exploit at best the available resources using them concurrently for different applications.

At the same time, such an operating way encounters also the demands of the users who want to pay just for the resources they use, and to avoid penalization in the completion time of their applications in the case that a scheduling policy different from FIFO were adopted.

3.3. The setup for cloud mapping problem

We assume to have an application subdivided into p tasks to be mapped on n nodes with $1 \leq n \leq p$ selected from the whole

number of cloud nodes N. Each node is identified by an integer value within the range $[1, N]$.

To determine efficient mapping solutions of TIG applications on cloud systems, alike other papers [25,45,46], it is indispensable to have information about the number of instructions α_i computed per time unit on each node i, and a good estimation of the communication bandwidth β_{ij} between any couple of nodes i and j.

In general, the runtime state of resources, such as CPU load and network capacity, is acquired either through statistical estimations in a given time span, or gathered by tracking periodically and forecasting dynamically resource conditions [47,48].

Since clouds address non-dedicated resources, their current workloads must be considered to correctly evaluate the computation time of the tasks. We indicate with $\ell_i(\Delta t)$ the average load of the node i at any given time span Δt with $\ell_i(\Delta t) \in [0.0, 1.0]$, where 0.0 means a node completely discharged and 1.0 a node loaded at 100%. Hence $(1 - \ell_i(\Delta t)) \cdot \alpha_i$ represents the power of the node i available for the execution of one or more tasks of the application to be mapped. Analogous considerations can be adduced to establish the available communication bandwidths among nodes. The current workload and the already employed bandwidth on each physical cloud node are provided by the *CRS* module, outlined in Section 3.1.

Besides raw resource information, application task properties are also necessary for making an effective deployment. We assume that we know for each task k the number of instructions γ_k and the amount of communications ψ_{km} to be executed between the kth and the mth task $\forall m \neq k$. Obviously, ψ_{km} is the generic element of a $p \cdot p$ symmetric matrix $\boldsymbol{\psi}$ with all null elements on the main diagonal. All this information can be obtained by means of some performance-modeling strategies proposed in literature for predicting execution time of parallel applications [49–52].

4. The algorithms

4.1. The TIG model

In this model a parallel application is represented by a Task Interaction Graph (TIG) $G(V, E)$, where $V = V_1, V_2, \ldots, V_n$ is the set of nodes and $E \subset V \times V$ is the set of the edges. The nodes represent the application tasks and the edges represent the mutual communication among the tasks. A function τ^{comp} gives the computation cost of tasks and a function τ^{comm} takes into account the communication cost for message passing on edges. A task V_i is a neighbor of a task V_j if $(V_i, V_j) \in E$. The relation is commutative in the sense that if $(V_i, V_j) \in E$ then $(V_j, V_i) \in E$.

The tasks exchange information during their execution but in this model the temporal dependences in the task execution are not explicitly addressed: all the tasks are considered simultaneously executable and communications can take place at any time during the computation. A more comprehensive description of the TIG model can be found in [19].

It is to note that all the parallel multitask applications, whose communications take place through explicit message passing, such as MPI and PVM, and with no precedence relationship among the tasks, fit in this model.

4.2. The cost function

The appearance of run-time dependences yields difficult to evaluate the execution time of a TIG application. Several cost functions have been proposed in literature which try to approximate the execution time expected for the application and simplify the real situation in order to perform an evaluation of the cost once fixed the heuristic. Thus, the mapping problem can be seen as an optimization problem by defining a cost function to be minimized. In general cost functions consider both computation and communication costs of each task in the TIG model but these costs may be handled in different ways. Basically, the cost functions used with the task-mapping problem may be categorized as belonging either to the minimax model [53] or to the summed cost model [18]. The summed cost method tries to balance the computation cost between all the nodes, i.e., by minimizing the load imbalance cost, while keeping the amount of intranode communications to a minimum, by minimizing the global communication cost.

Within this paper we make reference to the minimax model. In this model the computation and communication cost of each processor for a given mapping, known as resource use time, is estimated and the maximum cost among all the nodes is to be minimized.

Formally, let $\boldsymbol{\mu}$ be a solution consisting of an array of p integer values, where $\mu[i] = j$ means that the ith task of the application is placed on the jth node of the cloud ($1 \leq j \leq N$). Denoting with τ_{ij}^{comp} and τ_{ij}^{comm} respectively the computation and the communication times requested to execute the task i on the node j it is assigned to, the total resource time needed to execute i on j is:

$$\tau_{ij} = \tau_{ij}^{\text{comp}} + \tau_{ij}^{\text{comm}}.$$

Let τ_j^s be the summation of the resource times needed by all the tasks assigned to the jth node by the current mapping. This value is the time spent by node j in executing computations and communications of all the tasks assigned to it by the proposed solution. Clearly, τ_j^s is equal to zero for all the nodes not included in the vector $\boldsymbol{\mu}$.

Then, the quality of the proposed solution is computed by the function:

$$\Phi(\boldsymbol{\mu}) = \max_{j \in [1,N]} \{\tau_j^s\} \tag{1}$$

and, according to the minimax model, the goal is to achieve the smallest value among these maxima.

Our mapping approach follows this minimax model. In fact, the function *evaluate()*, mentioned in the forthcoming Section 4.4, computes exactly $\Phi(\boldsymbol{\mu})$. Nevertheless, some evaluations inside this function depend on the hypothesized cloud environment. In particular, τ_{ij} is evaluated on the basis of the computation power and of the bandwidth that remain available once deducted the current cloud workloads.

By using the notations introduced in Section 3.3, the values in the above equation can be computed respectively as: $\tau_{ij}^{\text{comp}} = \gamma_i / ((1 - \ell_j(\Delta t)) \cdot \alpha_j)$ and $\tau_{ij}^{\text{comm}} = \sum_{k=1, k \neq i}^{p} \psi_{ik} / \beta_{j\mu[k]}$.

4.3. The seminal algorithms

The Min–min heuristic works by taking into account the set U of the unmapped tasks. Initially this set contains all the p application tasks to be mapped.

The first step consists in finding for each task i its minimum completion time over all the N cloud nodes: $\tau_i^{\min} = min_{1 \leq j \leq N}\{\tau_{ij}\}$. Let us denote by M the set of the $\tau_i^{\min}$ for all the tasks in the parallel application.

The second step selects from M the task k with the *minimum* among these completion times $\tau_i^{\min}$, i.e., k is the task for which $\tau_k^{\min} = min_{i \in U}\{\tau_i^{\min}\}$, and assigns it to the corresponding node of the cloud that ensures that completion time. This choice of the minimum among the minima is the reason for naming this algorithm Min–min.

The third step, finally, removes the newly mapped task k from U.

The above steps two and three are repeated until all the tasks of the parallel application are mapped, i.e., exactly p times. At the end of these p iterations the set U will be empty.

```
Min–min–C algorithm
Require
Information on number, features, and estimated average local load of cloud nodes
Information on number and requests of parallel application tasks
begin
  for i=1 to p do task_mapped[i] = 0;
  for i=1 to p do μ[i] = −1;
  for i=1 to p do
    begin for
      (task_chosen, node_chosen) = select_task();
      μ[task_chosen] = node_chosen;
      task_mapped[task_chosen] = 1;
    endfor
  resource use time Φ(μ) = evaluate(solution);
end


function select_task()
begin
  for j=1 to N do β_j^aver = 1/N · Σ_{i=1,i≠j}^{N} β_ji;
  for each task i not allocated yet (i.e., such that task_mapped[i] = 0) do
    begin for
      for j=1 to N do
        begin for
          τ_ij^comp = γ_i/((1 − ℓ_j(Δt)) · α_j);
          τ_ij^comm = 0.0;
          for k=1 to p (k ≠ i) do
            begin for
              if
                (task_mapped[k] = 1)      // i.e., if k is already allocated
                  τ_ij^comm = τ_ij^comm + (ψ_ik/β_μ[i][μ[k]]);
                else                      // i.e., if k is not yet allocated on any node, (task_mapped[k] = 0)
                  τ_ij^comm = τ_ij^comm + (ψ_ik/β_μ[i]^aver);
            endfor
          τ_ij = τ_ij^comp + τ_ij^comm;
        endfor
      τ_i^min = min_{1≤j≤N}{τ_ij}
      node[i] = the node corresponding to τ_i^min;
    endfor
  τ_x^min = min_{i∈U}{τ_i^min}
  task_chosen = x;
  node_chosen = μ[x];
  return task_chosen and node_chosen;
end
```

Fig. 2. A pseudo-code description of the Min–min-C procedure.

The Max–min heuristic is very similar to Min–min, and also begins with the set U of all unmapped tasks.

The first step is absolutely identical as in the previous algorithm, and, as its result, for each task i in the parallel application its minimum completion time $\tau_i^{\min}$ is found.

In its second step, Max–min selects the task w with the overall *maximum* completion time from M, i.e., $\tau_w^{max} = \max_{i \in U}\{\tau_i^{\min}\}$, and assigns it to the corresponding node in the cloud. This choice of the maximum among the minima explains why this heuristic is called Max–min.

Thirdly, Max–min removes from U the newly mapped task.

This process iterates steps two and three for exactly p times. At the end there will be no tasks to be mapped, i.e., U will be empty.

These heuristics have proven useful when mapping independent tasks in heterogeneous environments. They can manage the concepts of node load, node computing power, and number of instructions for a task as described earlier.

4.4. The proposed algorithms

The algorithms we propose here are extensions to the above reported Min–min and Max–min heuristics. The basic idea is to add a way to take communications into account as well. Namely, the concepts of communication bandwidth and amount of communication for a task should be considered.

At a quite high descriptive level, both Min–min-C and Max–min-C start by considering all the p tasks of a parallel application as not mapped. Then, the algorithm calls for p times a function *select_task*, and receives from it two pieces of information: the task chosen and the node onto which map this task. The task chosen is removed from the set of unmapped tasks. At the end of the p iterations all the p tasks composing the application will be mapped.

Within the *select_task* function, when trying to map a not-yet-mapped task i onto a node j, they both compute exactly the communication time with each already mapped task k on a node x on the basis of the actual bandwidth β_{jx}. If the task k, instead, has not been mapped yet, the communication between i and k is supposed to take place at a bandwidth equal to the average value over all the possible bandwidths available for the node j with all the other nodes contained in the cloud, i.e., β_j^{aver}. Computations, of course, are taken into account in exactly the same way as in their originating algorithms, and computation and communication times are summed to achieve the execution time of task i onto node j.

At the end of the algorithm, of course, the resource use time for the achieved mapping will be computed in the *evaluate* function by taking into account for the communications all the exact bandwidths between the nodes onto which the application tasks have been allocated. This computation is described in full details in the previous Section 4.1.

A pseudo-code description for Min–min-C is shown in Fig. 2.

The pseudo-code for Max–min-C is the same, apart from one single difference: in the function *select_task* the node with the highest value of time is to be saved in node[i], rather than that with the lowest value as it is for Min–min-C.

Table 1
Power expressed in MIPS and node indices.

	A			B		C	D			E				F	
	A_1	A_2	A_3	B_1	B_2	C	D_1	D_2	D_3	E_1	E_2	E_3	E_4	F_1	F_2
α	3000	2000	1500	1400	1200	1000	900	800	700	600	500	400	300	200	100
n_I	1–12	13–28	29–36	37–46	47–62	63–94	95–98	99–118	119–134	135–142	143–158	159–168	169–188	189–208	209–216

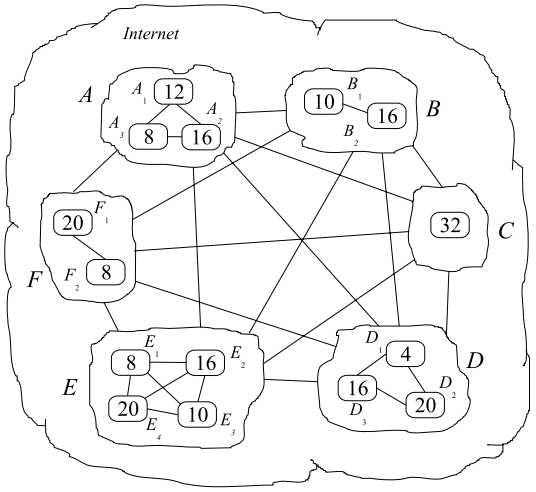

Fig. 3. The cloud architecture.

5. Experiments and results

The algorithms have been implemented in C language and all the experiments have been effected on a MacBookPro4.1 Intel Core Duo 2.4 GHz, 2 GB RAM. Less than one second is needed to execute all the four algorithms for each problem.

The experimental phase has been subdivided in two subsections, i.e. *Experiment A* and *Experiment B*. To carry out all the experiments we have hypothesized a cloud architecture made up of 216 nodes grouped into six sites, denoted with *A*, *B*, *C*, *D*, *E*, and *F* with 36, 26, 32, 40, 54, and 28 nodes respectively. This cloud structure is depicted in Fig. 3. As it can be seen, except *C*, each site consists of more subsites with different numbers of nodes. As an example *F* is subdivided into two subsites F_1 and F_2 made up of 20 and 8 nodes respectively. Each node is characterized by a set of resource properties.

Without loss of generality we suppose that all the nodes in the same subsite have the same nominal power α expressed in terms of millions of instructions per second (MIPS). For the sake of simplicity we have hypothesized for each node four communication typologies. The first is the bandwidth β_{ii} available when tasks are mapped on the same node (*intranode communication*); the second is the bandwidth β_{ij} between the nodes i and j belonging either to the same subsite (*intrasubsite communication*), to different subsites of the same site (*intrasite communication*) or to different sites (*intersite communication*). The intranode bandwidths β_{ii}, usually higher than β_{ij}, have all been fixed to 100 000 Mb/s.

The features of the nodes and the structures of the applications considered for the two experiments have been chosen with two different objectives. Within the *Experiment A* the test bed considered is very special and the objective is to simply evaluate the goodness of the mapping solution retrieved, as illustrated in the Section 5.1.2. Instead the *Experiment B* has been carried out in a more realistic cloud environment and with irregular TIG applications.

5.1. Experiment A

5.1.1. Cloud features

In Table 1 the computational capacities and the indices of the nodes are shown per each subsite of the cloud architecture depicted in Fig. 3. To give an example all the nodes of the subsite E_1 have $\alpha = 600$. It is to observe that the power decreases from 3000 to 100 MIPS when going from A_1 to F_2. Hereinafter we shall denote the nodes by means of the indices n_I still shown in Table 1, so that, for instance, 49 is the third node in B_2, while 138 is the fourth node in E_1.

Finally, the input and output bandwidths between two nodes are supposed to be equal. In our case the intersite, the intrasite, and the intrasubsite bandwidths are reported in Table 2. It is to remark that the bandwidth increases from 100 to 10 000 when going from A_1 to F_2, and the bandwidth between nodes belonging to the same site is supposed to be lower than two nodes of the same subsite. For example each node of D_2 communicates with a node of D_3 with a bandwidth of 75 Mb/s, while the bandwidth inside D_3 is 400 Mb/s.

5.1.2. Applications

To the best of our knowledge, no papers exist that provide mapping solutions for environments and TIG applications with features comparable with those hypothesized by us. In fact, most of those described in scientific literature make reference to the allocation of independent tasks only. Only a few of them consider applications with communicating tasks, but they take into account only DAGs [10–12], whereas we are in the most general case of TIGs. And even when the environments described by other authors could be taken into account, the further working conditions (average load and bandwidth use), not specified by them but crucial for us, make any comparison with their algorithms impossible. Therefore in the following no results from algorithms other than those described here are reported.

In the light of the above considerations, it has been conceived to take into account here TIG applications with regular topologies, and to arrange the experiments so as to attain solutions appreciable in absolute rather than relative value. To evaluate the goodness of our mapper on allocation problems, experimental environments for which the optimal solutions can be simply retrieved have been conceived. In fact, by choosing conveniently the computing capabilities, the communication bandwidths and the load conditions of all the cloud nodes, and given appropriate applications, it is possible to find out 'by hand' the related optimal mapping solutions. In these situations, the quality of the provided solutions can be rapidly verified by comparison with the expected optimal ones. If irregular TIG applications were chosen, a 'qualitative' analysis would be impracticable since the optimal or suboptimal solutions would be very difficult to deduct. This holds also for real applications for which the evaluation of quality of the mapping is arduous. Our aim has been to test the ability of the mapping tool to work in predetermined conditions, so that we can be confident that it works properly also in case of random loads.

Although the experiments have been carried out on different scenarios for cloud systems and applications in terms of load, number of tasks, computations, and communications, here the solutions obtained only for four different types of quite general application structures, which are described below, are reported.

Table 2
Intersite, intrasite, and intrasubsite bandwidths expressed in Mb/s.

Site	*A*			*B*		*C*	*D*			*E*				*F*	
	A_1	A_2	A_3	B_1	B_2	C	D_1	D_2	D_3	E_1	E_2	E_3	E_4	F_1	F_2
A_1	100														
A_2	50	100													
A_3	75	75	100												
B_1	4	4	4	200											
B_2	4	4	4	75	200										
C	6	6	6	6	6	300									
D_1	8	8	8	8	8	8	400								
D_2	8	8	8	8	8	8	50	400							
D_3	8	8	8	8	8	8	75	75	400						
E_1	10	10	10	10	10	10	10	10	10	800					
E_2	10	10	10	10	10	10	10	10	10	100	800				
E_3	10	10	10	10	10	10	10	10	10	200	200	800			
E_4	10	10	10	10	10	10	10	10	10	100	100	200	800		
F_1	16	16	16	16	16	16	16	16	16	16	16	16	16	1000	
F_2	16	16	16	16	16	16	16	16	16	16	16	16	16	400	10 000

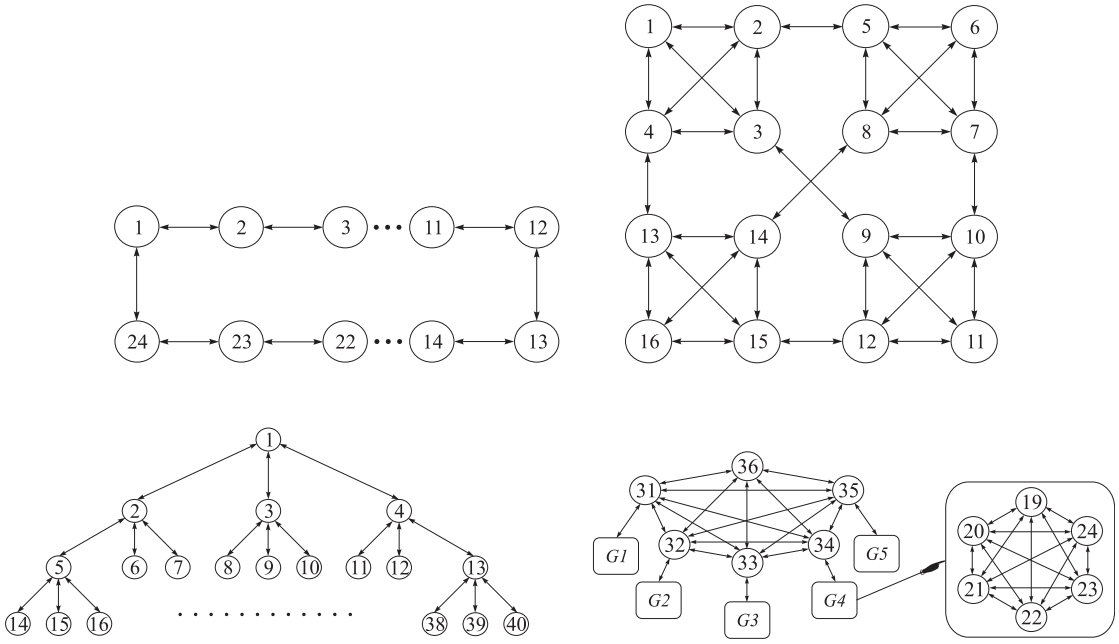

Fig. 4. The application structures faced in this paper. Top left: ring. Top right: WK. Bottom left: ternary tree. Bottom right: mesh.

The four application structures are reported in Fig. 4. The first structure has been considered as a TIG application made up of $p = 24$ tasks numbered from 1 to 24, and structured as a ring. The top-left pane of Fig. 4 illustrates the application structure. These tasks are divided into three groups $G1$, $G2$, and $G3$ constituted by 10 (1–10), 8 (11–18), and 6 (19–24) tasks respectively. The ten tasks in $G1$ have $\gamma_k = 90$ Giga Instructions (GI) to execute, those of $G2$ have $\gamma_k = 10$ GI while in $G3$ $\gamma_k = 1000$ Mega Instructions (MI). The consecutive tasks of $G1$ exchange 100 Mb each other while those consecutive of $G2$ and of $G3$ exchange one another 3000 and 100 000 Mb respectively. The ring is closed assuming that the first and the last tasks of two consecutive groups exchange 10 Mb.

The second structure has been designed as a TIG application composed of $p = 16$ tasks numbered from 1 to 16, and arranged in the WK-recursive topology [54] of degree 4 and level 2 (WK(4, 2)) shown in the top-right pane of Fig. 4. These tasks are divided into four groups $G1$, $G2$, $G3$, and $G4$ each consisting of four consecutively numbered tasks. As an example the $G1$ includes the tasks from 1 to 4 and so on. The amount of computation and communication of each task varies according to the group it belongs to, and to the position occupied in the topology. In particular as regards the computation, the tasks of $G1$, $G2$, $G3$, and $G4$ have to execute $\gamma_k = 160$ GI, $\gamma_k = 45$ GI, $\gamma_k = 15$ GI, and $\gamma_k = 900$ MI respectively. As the communication is concerned with, inside each of the four groups an all-to-all communication takes place (each task of each of the four groups exchanges data with each of the other three tasks in the same group). The amount of this exchange is 100, 1000, 10 000, and 100 000 Mb going from the tasks belonging to $G1$ to those of $G4$. In addition, there is an inter-group communication of 100 Mb which occurs by means of the links connecting the task pairs (2,), (7, 10), (12, 15), (13, 4), (3, 9), and (8, 14).

The third structure has been imagined as a TIG application composed of $p = 40$ tasks numbered from 1 to 40, and structured in a ternary tree as depicted in the bottom-left pane of Fig. 4 with task 1 in the root node at the top level, tasks from 2 to 4 in the first level, and so on. The amount of computation and communication that each task has to perform depends on the level it is placed on. Task 1 executes 100 MI and exchanges 10 000 Mb with each of its three leaves. Each task belonging to the first level, in addition to exchanging 10 000 Mb with the root node, executes 1 GI and exchanges 1000 Mb with each of its three leaves. Each task of the second level exchanges 1000 Mb with its vertex, executes 10 GI, and exchanges 100 Mb with its own leaves. Finally, each of the 27 leaves of the third level executes 100 GI and exchanges 100 Mb with its vertex.

Table 3
Numerical results achieved by the algorithms: resource utilization times (in seconds) for the TIG applications.

Problem	Min–min	Max–min	Min–min-C	Max–min-C
ring-*no-load*	2065.766	2049.549	68.625	76.138
ring-*load*	2112.500	2169.816	92.002	156.102
ring-*random*	504.860	2003.531	493.101	493.101
WK-*no-load*	3058.633	3002.450	68.370	68.370
WK-*load*	4992.992	4674.506	249.047	287.967
WK-*random*	456.417	4519.258	461.356	461.356
tree-*no-load*	3050.700	2111.302	858.333	930.653
tree-*load*	3259.285	7617.710	2182.593	2486.846
tree-*random*	1406.810	4377.314	1406.810	1961.576
mesh-*no-load*	5609.466	8722.199	201.500	201.500
mesh-*load*	21053.200	9993.614	1220.335	1105.268
mesh-*random*	1867.181	9972.520	1837.153	1837.153

The fourth structure has been conceived as a TIG application made up of $p = 36$ tasks numbered from 1 to 36, and arranged on a mesh as outlined in the bottom-right pane of Fig. 4. The task 36 in the root node at the top level is linked to the five different tasks of the first level. Each of these five tasks is connected to a group of six tasks characterized by all-to-all communication. These five groups are indicated as $G1$, $G2$, $G3$, $G4$, and $G5$, and represent the second level. As an example, the 'blast' of $G4$ contents is shown. The root task executes 1 GI and exchanges 10 000 Mb with each of its five leaves. Each task belonging to the first level, in addition to exchanging 10 000 Mb with the root node, executes 10 GI, exchanges 10 Mb with each of the six tasks of the group to which is connected, and 10 000 Mb with each of the other four tasks of its level. The amount of computation and communication carried out by each task depends on the group it belongs to. In particular, each task of $G1$ executes 300 GI and exchanges 10 Mb, of $G2$ carries out 100 GI and exchanges 3000 Mb, of $G3$ executes 50 GI and exchanges 5000 Mb, of $G4$ executes 25 GI and exchanges 10 000 Mb, and finally each task of $G5$ performs 1000 MI and exchanges 100 000 Mb.

For each such general structure, three different experiments have been performed in different operating node conditions, i.e., *no-load*, *load*, and *random* cases.

In the first, all the nodes have been supposed unloaded, i.e., $\ell_i(\Delta t) = 0.0$, $1 \leq i \leq n$ (*no-load* case).

In the second, loads have been chosen for each node in a controlled way, differing from a structure to another. This means that most nodes still have $\ell_i(\Delta t) = 0.0$, apart from a few that are supposed loaded (*load* case). Namely, for the ring, the loads of the nodes in the range [1, 20] have $\ell_i(\Delta t) = 0.7$. For the WK, the nodes within the interval [1, 10] have $\ell_i(\Delta t) = 0.7$. For the ternary tree, the nodes in the range [1, 30] have $\ell_i(\Delta t) = 0.9$. Finally, for the mesh, the nodes within [1, 34] have been supposed loaded with $\ell_i(\Delta t) = 0.7$. In all the four structures these modifications in loads have been taken because some of the now loaded nodes are chosen in the solutions of the first experiment. The aim is to see if the mappers realize that now better solutions, different from the former ones, exist.

In the third, and last experiment, once established that the system works properly in predetermined conditions, we can test it by randomly choosing $\ell_i(\Delta t)$ for each node within the range [0.2, 0.7] (*random* case).

Thus, a total of twelve different test cases has been designed.

5.1.3. Results

In Table 3 the results achieved by the four tested algorithms over the twelve different test cases are shown. In particular the values in the table represent the estimated resource use times (in seconds) of the different TIG applications on the basis of the algorithm used to perform the mapping.

It is evident that both Min–min-C and Max–min-C outperform their corresponding versions Min–min and Max–min. On the ring-*no-load* and WK-*no-load* problems, for example, the two former achieve results better by two order of magnitude than their two originating heuristics. In most cases the improvement is of one order of magnitude. Min–min-C seems to perform slightly better than Max–min-C since it achieves the best results over ten test cases (five times absolute best, five times a draw), whereas the latter wins in five test cases (once absolute best, four times a draw). The original Min–min performs as the best heuristic in two cases (once best, once a draw). Finally, never is Max–min the best performing algorithm.

Furthermore, closer examination of Table 3 reveals that Min–min-C is always either the first or the second technique, and Max–min-C does so too, apart from one case in which it ranks as third. Max–min is in all cases either the third of the fourth method, and Min–min also is in the same range apart from two cases in which it is the best.

As long as the *no-load* cases are faced, the superiority of Min–min-C and Max–min-C is evident by one or two orders of magnitude. When some nodes get loaded, the execution times found by all the four methods of course increase, and there is still a gap of at least one order of magnitude between the two couples, apart from the ternary tree structure. Finally, when the *random–load* situations are investigated, Min–min is able to reduce this gap and becomes comparable, whereas performance of Max–min remains worse by about one order of magnitude.

As an example of the solutions found by the four algorithms, let us take the WK-*no-load* problem into account. The four solutions in terms of resource utilization time are:

Min–min: $\boldsymbol{\mu} = \{1\ 2\ 3\ 4\ 9\ 10\ 11\ 12\ 5\ 6\ 7\ 8\ 1\ 2\ 3\ 4\}$
$\Phi(\boldsymbol{\mu}) = 3058.633$ s
Max–min: $\boldsymbol{\mu} = \{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\}$
$\Phi(\boldsymbol{\mu}) = 3002.450$ s
Min–min-C: $\boldsymbol{\mu} = \{2\ 5\ 3\ 4\ 1\ 1\ 1\ 1\ 6\ 6\ 6\ 6\ 209\ 209\ 209\ 209\}$
$\Phi(\boldsymbol{\mu}) = 68.370$ s
Max–min-C: $\boldsymbol{\mu} = \{5\ 3\ 4\ 2\ 6\ 6\ 6\ 6\ 1\ 1\ 1\ 1\ 209\ 209\ 209\ 209\}$
$\Phi(\boldsymbol{\mu}) = 68.370$ s.

According to the encoding reported in Section 4.1, for instance the solution related to Max–min-C maps the task 1 on node 5, the task 2 on node 3, the task 3 on node 4, the task 4 on node 2 and so on until to the task 16 that is deployed on node 209.

It is very interesting to note that the solution found by both Min–min-C and Max–min-C is a suboptimal one. In fact, it allocates the four groups of tasks on nodes of A_1, A_1, A_1, and F_2 respectively. The nodes of A_1 are the most powerful ones, so they are well suited to the heavy-computing tasks of the first group $G1$. Only one task is mapped onto any such node, so as not to overload those nodes. The tasks of the groups $G2$ and $G3$, instead, are quite balanced as regards computation and communication. The four tasks of $G2$ have been mapped on a same powerful node belonging to A_1 (node 1 for Min–min-C and node 6 for Max–min-C). In this way, the time required for the communications has been highly decreased, as that for the computations. A similar decision has been taken for the tasks belonging to the group $G3$, all mapped on a powerful node of A_1, i.e., 6 for Min–min-C and 1 for Max–min-C. The tasks in the group $G4$, instead, are the most heavily communicating ones, and have been deployed onto one single node of F_2, so that the time needed for the communications among them can be kept low. Furthermore, using a node of F_2 is sensible because this group $G4$ has to communicate with the other groups too, and F_2 has the highest intrasubsite bandwidth.

Table 4
Power expressed in MIPS and node indices.

	A			B		C	D			E				F	
	A_1	A_2	A_3	B_1	B_2	C	D_1	D_2	D_3	E_1	E_2	E_3	E_4	F_1	F_2
α	2000	6400	57 063	140564	6400	2000	12 096	27 079	76 383	68 200	147 600	6400	12 000	18 938	68 200
n_l	1–12	13–28	29–36	37–46	47–62	63–94	95–98	99–118	119–134	135–142	143–158	159–168	169–188	189–208	209–216

Table 5
Intersite, intrasite, and intrasubsite bandwidths expressed in Mb/s.

Site	A			B		C	D			E				F	
	A_1	A_2	A_3	B_1	B_2	C	D_1	D_2	D_3	E_1	E_2	E_3	E_4	F_1	F_2
A_1	1000														
A_2	100	1000													
A_3	100	100	10000												
B_1	20	20	20	1000											
B_2	20	20	20	100	800										
C	34	34	34	16	16	800									
D_1	10	10	10	10	10	20	1000								
D_2	10	10	10	10	10	20	100	1000							
D_3	10	10	10	10	10	20	100	100	1000						
E_1	10	10	10	10	10	10	10	10	10	100					
E_2	10	10	10	10	10	10	10	10	10	100	100				
E_3	10	10	10	10	10	10	10	10	10	100	100	1000			
E_4	10	10	10	10	10	10	10	10	10	100	100	100	1000		
F_1	16	16	16	16	16	16	16	16	16	16	16	16	16	1000	
F_2	16	16	16	16	16	16	16	16	16	16	16	16	16	100	1000

The two original algorithms Min–min and Max–min, instead, require very high times, because map more tasks of different types on a same node, so that they can take advantage of the computational power of nodes in A_1. Nevertheless these algorithms have to suffer from their slowness in communications since an optimization of the deployment of the communication-intensive tasks is not taken into account.

It is interesting to note that, when loads are added to the nodes in the range [1, 10], as it is in the second experiment on the WK structure, all the four mappers have correctly avoided using the now loaded and therefore less powerful nodes in [1, 10], and have turned their attention to the now most powerful nodes, that are the only two in A_1 that have no load, i.e., 11 and 12.

In fact, for this experiment the new best solutions become:

Min–min:

$$\boldsymbol{\mu} = \{11\ 12\ 11\ 12\ 11\ 11\ 11\ 11\ 11\ 12\ 11\ 11\ 11\ 11\ 12\ 11\ 12\}$$
$$\Phi(\boldsymbol{\mu}) = 4992.992 \text{ s}$$

Max–min:

$$\boldsymbol{\mu} = \{11\ 12\ 11\ 11\ 11\ 12\ 12\ 12\ 12\ 11\ 11\ 11\ 11\ 11\ 12\ 12\}$$
$$\Phi(\boldsymbol{\mu}) = 4674.506 \text{ s}$$

Min–min-C:

$$\boldsymbol{\mu} = \{12\ 12\ 12\ 12\ 11\ 11\ 11\ 11\ 12\ 12\ 12\ 12\ 209\ 209\ 209\ 209\}$$
$$\Phi(\boldsymbol{\mu}) = 249.047 \text{ s}$$

Max–min-C:

$$\boldsymbol{\mu} = \{12\ 12\ 12\ 12\ 12\ 12\ 12\ 12\ 11\ 11\ 11\ 11\ 209\ 209\ 209\ 209\}$$
$$\Phi(\boldsymbol{\mu}) = 287.967 \text{ s}.$$

Also in this case, as in the previous experiment, the superiority of the mapping performed by Min–min-C and Max–min-C with respect to Min–min and Max–min lies in the fact that the former mappers place the most heavily communicating tasks on a node well suited to communications. The latter mappers, instead, overload the most computation-bound nodes, resulting in bad allocation, incapable of efficiently dealing with communications, and in a much higher time for the whole application.

5.2. Experiment B

5.2.1. Cloud features, applications and results

Leaving the structure of the cloud unchanged, we have here supposed the computational capacities and the indices of the nodes as reported in Table 4. These computational powers are typical of real processors and range from 2000 MIPS of the ARM Cortex A8 processor (2005) to 147 600 MIPS of the Intel Core i7-980X Extreme Edition processor (2010).

We have considered as communication bandwidth the common values shown in Table 5, and have randomly set the load $\ell_i(\Delta t)$ of each cloud node in the range [0.1, 0.5].

Moreover, we have assumed to have irregular TIG applications characterized by a number of tasks p which varies randomly within the range [10, 50]. For each task k the number of instructions γ_k is randomly set within 10 GI and 1000 GI, so as the amount of communications ψ_{km} to be exchanged between the kth and the mth task $\forall m \neq k$ are randomly chosen within 100 Mb and 10 Gb. Each task communicates with a number of partners randomly chosen within 1 and $\lfloor p/3 \rfloor$ and the partners are selected still randomly within 1 and p.

By doing so, we have generated 15 different instances of TIGs. It should be noted that on each problem Min–min-C achieves better result than Min–min, and so does Max–min-C with respect to Max–min. The results obtained by the different algorithms are outlined in Table 6.

5.2.2. Statistical analysis

To compare the algorithms, a classical statistical approach based on nonparametric statistical tests has been carried out, following [55,56]. Therefore, multiple comparison analysis has been effected by means of Friedman, Aligned Friedman, and Quade tests. To do so, the ControlTest package [57] has been used. It is a Java package freely downloadable at *http://sci2s.ugr.es/sicidm/* [58], developed to compute the rankings for these tests, and to carry out the related post-hoc procedures and the computation of the adjusted p–values. A brief explanation of these statistical tests is reported in the Appendix.

The results for the one-to-all analysis are reported in the following.

Table 6
Resource utilization times (in seconds) for the instances of the TIG applications.

Instance	p	Min–min	Max–min	Min–min-C	Max–min-C
Inst1	14	239.279	61.367	61.329	61.079
Inst2	25	249.754	2156.842	74.327	67.765
Inst3	42	4180.514	9223.947	193.866	202.596
Inst4	38	2123.789	4801.762	184.781	112.927
Inst5	26	201.605	3168.008	94.636	87.391
Inst6	31	321.101	2632.391	90.421	84.429
Inst7	44	3825.445	6603.204	252.092	198.753
Inst8	18	352.626	62.493	68.196	61.775
Inst9	29	411.255	4072.849	97.877	91.132
Inst10	37	2390.776	3622.805	156.163	101.317
Inst11	17	246.016	59.987	62.790	59.077
Inst12	20	309.050	2170.711	74.982	69.933
Inst13	46	4756.522	6462.253	229.050	259.180
Inst14	32	427.828	3072.310	85.606	80.155
Inst15	50	3971.087	7012.009	289.640	290.579

Table 7
Average rankings of the algorithms.

Algorithm	Friedman	Aligned Friedman	Quade
Max–min-C	1.250	18.375	1.331
Min–min-C	2.000	19.313	1.735
Min–min	3.250	39.500	3.074
Max–min	3.500	52.813	3.860
Statistic	32.400	12.286	57.451
p–value	0.000	0.007	0.000

Table 7 contains the results of the Friedman, Aligned Friedman, and Quade tests in terms of average rankings obtained by all the algorithms. The last two rows show the statistic and the p–value for each test, respectively. For Friedman and Aligned Friedman tests the statistic is distributed according to chi-square with 3 degrees of freedom, whereas for Quade test it is distributed according to F-distribution with 3 and 45 degrees of freedom.

In each of the three tests, the lower the value for an algorithm, the better the algorithm is. Max–min-C turns out to be the best in all of the three tests, and Min–min-C is always the second best heuristic. For all the tests both the versions taking communications into account largely outperform the classical algorithms. Min–min performs, on average, better than Max–min, as it was pointed out also in [25], where inter-task communications were not taken into account.

Furthermore, with the aim to examine if some hypotheses of equivalence between the best performing algorithm and the other ones can be rejected, the complete statistical analysis based on the post-hoc procedures ideated by Bonferroni, Holm, Holland, Rom, Finner, and Li has been carried out following [56]. Moreover, the adjusted p–values have been computed by means of [57].

Tables 8–10 report the results of this analysis performed at a level of significance $\alpha = 0.05$. The level of significance represents the maximum allowable probability of incorrectly rejecting a given null hypothesis. In our case, as an example, if it is equal to 0.05, this means that if an equivalence hypothesis is rejected, there is a 5% probability of making a mistake in rejecting it, so a 95% that we are correctly rejecting it.

In these tables the other algorithms are ranked in terms of distance from the best performing one, i.e., Max–min-C in all the three cases, and each algorithm is compared against this latter with the aim to investigate whether or not the equivalence hypothesis can be rejected. For each algorithm each table reports the z value, the unadjusted p–value, and the adjusted p–values according to the different post-hoc procedures. The variable z represents the test statistic for comparing the algorithms, and its definition depends on the main nonparametric test used. In [56] all the different definitions for z, corresponding to the different tests, are reported. The last row in the tables contains for each procedure the threshold value *Th* such that the procedure considered rejects those equivalence hypotheses that have an adjusted p–value lower than or equal to *Th*.

Summarizing the results of these tables, Max–min-C is statistically superior to both Max–min and Min–min. As concerns its comparison against Min–min-C, for all the three tests there are three post-hoc procedures, i.e., those by Bonferroni, Rom, and Li, that cannot reject the equivalence hypothesis, whereas the other three tests by Holm, Holland, and Finner reject this hypothesis.

Finally, Table 11 shows the pairwise comparison between these four algorithms. Two post-hoc procedures, i.e., those by Holm and Shaffer, are considered here. Only for Min–min and Max–min cannot the equivalence hypothesis be rejected by both procedures. Furthermore, the equivalence between Min–min-C and Max–min-C can be rejected by Holm, whereas it cannot by Shaffer. All the other equivalence hypotheses can be statistically rejected by both procedures.

The conclusion of these pairwise comparisons is that the performance of Min–min-C and Max–min-C is statistically better than that obtained by the seminal versions of the respective algorithms, i.e., Min–min and Max–min.

6. Conclusions and future works

Even if not specifically designed to this aim, cloud platforms yield available the computational power needed to efficiently execute small and medium HPC applications.

This paper focuses on the task/node mapping problem of parallel multitask applications, modeled as TIGs, on non-dedicated cloud nodes. Two new resource-oriented mapping heuristics, Min–min-C and Max–min-C, have been introduced. They are very simple and their execution times are extremely low, and can provide users with solutions in a quasi-real time.

Table 8
Results of post-hoc procedures for Friedman test over all tools (at $\alpha = 0.05$).

i	Algorithm	p	$z = (R_0 - R_i)/SE$	Bonferroni	Holm	Holland	Rom	Finner	Li
3	Max–min	0.000	4.930	0.000	0.017	0.017	0.017	0.017	0.047
2	Min–min	0.000	4.382	0.000	0.025	0.025	0.025	0.034	0.047
1	Min–min-C	0.100	1.643	0.100	0.050	0.050	0.050	0.050	0.050
Th				0.017	0.050	0.050	0.025	0.050	0.047

Table 9
Results of post-hoc procedures for Aligned Friedman test over all tools (at $\alpha = 0.05$).

i	Algorithm	p	$z = (R_0 - R_i)/SE$	Bonferroni	Holm	Holland	Rom	Finner	Li
3	Max–min	0.000	5.231	0.000	0.017	0.017	0.017	0.017	0.006
2	Min–min	0.001	3.209	0.001	0.025	0.025	0.025	0.034	0.006
1	Min–min-C	0.887	0.142	0.887	0.050	0.050	0.050	0.050	0.050
Th				0.017	0.050	0.050	0.025	0.050	0.006

Table 10
Results of post-hoc procedures for Quade test over all tools (at $\alpha = 0.05$).

i	Algorithm	p	$z = (R_0 - R_i)/SE$	Bonferroni	Holm	Holland	Rom	Finner	Li
3	Max–min	0.000	3.977	0.000	0.017	0.017	0.017	0.017	0.025
2	Min–min	0.006	2.740	0.006	0.025	0.025	0.025	0.034	0.025
1	Min–min-C	0.525	0.636	0.525	0.050	0.050	0.050	0.050	0.050
Th				0.017	0.050	0.050	0.025	0.050	0.025

Table 11
Pairwise comparison between the algorithms.

Algorithm	**p**	**z**	Holm	Shaffer
Max–min vs. Max–min-C	0.000	4.930	0.008	0.008
Min–min vs. Max–min-C	0.000	4.382	0.010	0.017
Max–min vs. Min–min-C	0.001	3.286	0.013	0.017
Min–min vs. Min–min-C	0.006	2.739	0.017	0.017
Min–min-C vs. Max–min-C	0.100	1.643	0.025	0.025
Min–min vs. Max–min	0.584	0.548	0.050	0.050
Th			0.025	0.017

Experimental results reveal that these two new algorithms are an effective approach to efficiently face the mapping of MPI applications in terms of maximal resource utilization in cloud environments. Moreover, a statistical analysis evidences their superiority with respect to the two originating heuristics.

To properly work, our mapping strategies require the co-scheduling of all the tasks of the same application. At first sight, the co-scheduling might seem a limitation for the actual usability of our mappers, since currently this feature is not available in cloud, yet it is not an unrealistic hypothesis. In fact, it should be pointed out that very recently both academic [59] and industrial [60,61] research are aiming at implementing co-scheduling in clouds. This would yield a cloud model in which our mapper perfectly fits.

An interesting issue to investigate in our future activity is the choice of the next task to allocate when sequentially creating the solution. We think that this choice should consider for each task, apart from the currently accounted amount of computation, also the amount of communications and the number of partners involved in those data exchanges. Moreover, other heuristic algorithms used to map independent tasks only could be improved through the same idea used here and tested against communicating applications.

Appendix

A very basic explanation of the statistical tests reported in Section 5.2.2 is given here, for a case in which n_p problems and n_A algorithms are considered.

Friedman test ranks the algorithms for each problem separately; the best performing algorithm is assigned the rank of 1, the second best rank 2, and so on up to n_A. The ranks for each algorithm are summed over the n_p faced problems, and the sum is divided by n_p.

A drawback of the ranking scheme employed by the Friedman test is that when the number of algorithms for comparison is small, this may pose a disadvantage, and comparability among problems is desirable.

To overcome this problem, in Aligned Friedman test a value of location is computed as the average performance achieved by all algorithms in each problem. Then, the difference between the performance obtained by an algorithm and the value of location is obtained. This step is repeated for each combination of algorithms and problems. The resulting differences are then ranked from 1 to $n_A \cdot n_p$ relative to each other. Again, the ranks for each algorithm are summed over the faced problems, and the sum is divided by the number of problems.

The main drawback with the two above described tests is that they consider all problems to be equal in terms of importance.

Quade test takes into account the fact that some problems are more difficult or that the differences registered on the run of various algorithms over them are larger. Therefore, ranks are assigned to the problems themselves according to the width of the results range in each problem. Thus, the problem with the smallest range is assigned rank 1, the second smallest rank 2, and so on to the problem with the largest range, which gets rank n_p. So, each problem i is given a rank Q_i. Next, for each problem the product S_i^j is computed as: $S_i^j = Q_i \cdot (r_i^j - (n_A + 1)/2)$, where r_i^j is the rank of algorithm j within problem i. By doing so, this value takes into account both the relative importance of each observation within the problem, and the relative significance of the problem the observation refers to. Finally, a value S_j computed as $S_j = \sum_{i=1}^{n_p} S_i^j$, for $j = 1, 2, \ldots, n_A$ can be assigned to each algorithm.

References

[1] I. Foster, Z. Yong, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Proc. Workshop on Grid Computing Environments, IEEE Press, Austin, Texas, 2008, pp. 1–10.

[2] R. Buyya, C.S. Yeo, S. Venugopal, Market–oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities, Keynote paper, in: Proc. 10th Int. Conf. on High Performance Computing and Communications, IEEE Press, Dalian, China, 2008, pp. 25–27.

[3] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (6) (2009) 599–616.

[4] (2011, Jun) Google App Engine (Online). Available: https://developers.google.com/appengine/. Last checked December 6, 2013.

[5] (2006) Amazon Elastic Compute Cloud (Online). Available: http://aws.amazon.com/ec2/. Last checked December 6, 2013.

[6] Windows Azure (Online). Available: http://www.windowsazure.com (in Italian). Last checked December 6, 2013.

[7] J. Ekanayake, X. Qiu, T. Gunarathne, S. Beason, G. Fox, High performance parallel computing with clouds and cloud technologies, Technical Report, August 25 2009. (Online). Available: http://grids.ucs.indiana.edu/ptliupages/publications/CGLCloudReview.pdf.

[8] O. Niehörster, A. Brinkmann, G. Fels, J. Krüger, J. Simons, Enforcing SLAs in Scientific Clouds, in: Proceedings of IEEE International Conference on Cluster Computing, Heraklion, Crete, Greece, 2010, pp. 178–187.

[9] A. Gupta, L.V. Kale, F. Gioachin, V. March, C.H. Suen, B-S. Lee, P. Faraboschi, R. Kaufmann, D. Milojicic, Exploring the Performance and Mapping of HPC Applications to Platforms in the Cloud, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, Delft, The Netherlands, 2012, pp. 121–122.

[10] M. Iverson, F. Özgüner, Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment, in: Proc. 7th Heterogeneous Computing Workshop, Orlando, Florida, USA, 1998, pp. 70–78.

[11] R. Sakellariou, H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems, in: Proc. 13th Heterogeneous Computing Workshop, IEEE Press, Santa Fe, New Mexico, USA, 2004.

[12] G.Q. Liu, K.L. Poh, M. Xie, Iterative list scheduling for heterogeneous computing, J. Parallel Distrib. Comput. 65 (2005) 654–665.

[13] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, Z. Gu, Online optimization for scheduling preemptable tasks on IaaS cloud systems, J. Parallel Distrib. Comput. 72 (2012) 666–677.

[14] H. Kasahara, S. Narita, Practical multiprocessor scheduling algorithms for efficient parallel processing, IEEE Trans. Comput. 33 (11) (1984) 1023–1029.

[15] A.K.M. Khaled Ahsan Talukder, M. Kirley, R. Buyya, Multiobjective differential evolution for workflow execution on grids, in: Proc. 5th Int. Workshop Middleware for Grid Computing, ACM Press, Newport Beach, CA, 2007.

[16] F. Chan, J. Cao, Y. Sun, Graph scaling: a technique for automating program construction and deployment in cluster GOP, in: Proc. 5th Int. Workshop in Advanced Parallel Processing Technologies, in: Lecture Notes in Computer Science, vol. 2834, Springer, Berlin Heidelberg, Germany, 2003, pp. 254–264.
[17] D.L. Long, L.A. Clarke, Task interaction graphs for concurrency analysis, in: Proc. 11th Int. Conf. Software Engineering, ACM Press, Pittsburgh, Pennsylvania, USA, 1989, pp. 44–52.
[18] P. Sadayappan, F. Ercal, J. Ramanujam, Cluster partitioning approaches to mapping parallel programs onto a hypercube, Parallel Comput. 13 (1990) 1–16.
[19] L. Hluchy, M. Senar, M. Dobrucky, T.D. Viet, A. Ripoli, A. Cortes, Mapping and scheduling of parallel programs, in: J.C. Cunha, P. Kacsuk, S.C. Winter (Eds.), Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments, Advances in Computation: Theory and Practice, vol. 5, Nova Press, 2001, pp. 45–68. (chapter 3).
[20] S. Pandey, L. Wu, S. Guru, R. Buyya, A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, in: Proc. 24th Int. Conf. on Advanced Information Networking and Applications, IEEE Press, Perth, Australia, 2010, pp. 400–407.
[21] Y. Ge, G. Wei, GA-based task scheduler for the cloud computing systems, in: Proc. Int. Conf. on Web Information Systems and Mining, IEEE Press, Sanya, China, 2010, pp. 181–186.
[22] G. Gan, T. Huang, S. Gao, Genetic simulated annealing algorithm for task scheduling based on cloud computing environment, in: Proc. Int. Conf. on Intelligent Computing and Integrated Systems, IEEE Press, Gandhinagar Gujarat, India, 2010, pp. 60–63.
[23] Y. Fang, F. Wang, J. Ge, A task scheduling algorithm based on load balancing in cloud computing, in: Proc. Int. Conf. on Web Information Systems and Mining, IEEE Press, Sanya, China, 2010, pp. 271–277.
[24] N.A. Mehdi, A. Mamat, H. Ibrahim, S.K. Subramaniam, Impatient task mapping in elastic cloud using genetic algorithm, J. Comput. Sci. 7 (6) (2011) 877–883.
[25] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[26] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non identical processors, J. Assoc. Comput. Mach. 24 (2) (1977) 280–289.
[27] F. Dong, S.G. Akl, Scheduling Algorithms for Grid Computing: State of the Art and open Problems, School of Computing, Queen's University Kingston, Ontario, Tech. Rep. no. 2006–504, 2006.
[28] P. Luo, K. Lü, Z. Shi, A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 67 (2007) 695–714.
[29] S. Childs, B. Coghlan, J. Walsh, D. O'Callaghan, G. Quigley, E. Kenny, A virtual testGrid, or how to replicate a national grid, in: Proc. 15th Int. Symp. on High Performance Distributed Computing, Paris, France, 2006.
[30] B. Quétier, M. Jan, F. Cappello, One step further in large-scale evaluations: the V-DS environment, Research Report RR-6365, Institut National de Recherche en Informatique et en Automatique, 2007.
[31] C. Kleineweber, A. Keller, O. Niehörster, A. Brinkmann, Rule-based mapping of virtual machines in clouds, in: Proc. 19th Int. Euromicro Conference on Parallel, Distributed and Network-Based Processing, IEEE Press, Ayia Napa, Cyprus, 2011, pp. 527–534.
[32] A.E. Ezugwu, S.M. Buhari, S.B. Junaidu, Virtual machine allocation in cloud computing environment, Int. J. Cloud Appl. Comput. 3 (2) (2013) 47–60.
[33] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Softw. Eng. 15 (11) (1989) 1427–1436.
[34] F. Berman, High-performance schedulers, in: I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, San Francisco, CA, 1998, pp. 279–307.
[35] C.-C. Hui, S.T. Chanson, Allocating task interaction graphs to processors in heterogeneous networks, IEEE Trans. Parallel Distrib. Syst. 8 (9) (1997) 908–925.
[36] C. Roig, A. Ripoll, M.A. Senar, F. Guirado, E. Luque, A new model for static mapping of parallel applications with task and data parallelism, in: Proc. of the Int. Parallel and Distributed Processing Symposium, IEEE Press, Fort Lauderdale, Florida, USA, 2002.
[37] A. Jain, S. Sanyal, S.K. Das, R. Biswas, FastMap: a distributed scheme for mapping large scale applications onto computational grids, in: Proc. 2nd Int. Workshop on Challenges of Large Applications in Distributed Environments, IEEE Press, Arlington, TX, USA, 2004, pp. 118–127.
[38] R. Baraglia, R. Ferrini, L. Ricci, N. Tonellotto, R. Yahyapour, A launch-time scheduling heuristic for parallel application on wide area networks, J. Grid Comput. 6 (2008) 159–175.
[39] C. Vecchiola, S. Pandey, R. Buyya, High-performance cloud computing: a view of scientific applications, in: Proc. Int. Symp. on Pervasive Systems, Algorithms, and Networks, IEEE Press, Kaohsiung, Taiwan, 2009, pp. 4–16.
[40] D. Laforenza, Grid programming: some indications where we are headed, Parallel Comput. 28 (12) (2002) 1733–1752.
[41] A. Doğan, F. Özgüner, Scheduling of a meta-task with QoS requirements in heterogeneous computing systems, J. Parallel Distrib. Comput. 66 (2) (2006) 181–186.
[42] G. Mateescu, Quality of service on the grid via metascheduling with resource co-scheduling and co-reservation, Int. J. High Perform. Comput. Appl. 17 (3) (2003) 209–218.
[43] C. Qu, A grid advance reservation framework for co-allocation and co-reservation across heterogeneous local resource management systems, in: R. Wyrzykowski, et al. (Eds.), Proc. 7th Int. Conf. Parallel Processing and Applied Mathematics, in: Lecture Notes in Computer Science, vol. 4967, Springer, Berlin Heidelberg, Germany, 2007, pp. 770–779.
[44] E. Elmroth, J. Tordsson, Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions, Future Gener. Comput. Syst. 24 (6) (2008) 585–593.
[45] K. Ki-Hyung, H. Sang-Ryoul, Mapping cooperating grid applications by affinity for resource characteristics, in: Lecture Notes in Computer Science, vol. 3397, Springer, Berlin Heidelberg, Germany, 2005, pp. 313–322.
[46] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurr. 6 (3) (1998) 42–51.
[47] S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high-performance distributed computations, in: Proc. 6th Symp. High Performance Distributed Computing, IEEE Press, Portland, OR, USA, 1997, pp. 365–375.
[48] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proc. 10th Symp. High Performance Distributed Computing, IEEE Press, San Francisco, CA, 2001, pp. 181–194.
[49] J.M. Schopf, F. Berman, Using stochastic information to predict application behavior on contended resources, Internat. J. Found Comput. Sci. 12 (3) (2001) 341–364.
[50] V. Taylor, X. Wu, R. Stevens, Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications, AGM SIGMETRICS Perform. Eval. Rev. 30 (4) (2004) 13–18.
[51] H.A. Sanjay, S. Vadhiyar, Performance modeling of parallel applications for grid scheduling, J. Parallel Distrib. Comput. 68 (2008) 1135–1145.
[52] V. Adve, M. Vernon, Parallel program performance prediction using deterministic graph analysis, ACM Trans. Comput. Syst. 22 (1) (2004) 94–136.
[53] V.M. Lo, Heuristic algorithms for task assignment in distributed systems, IEEE Trans. Comput. 37 (11) (1988) 1384–1397.
[54] G. Della Vecchia, C. Sanges, A recursively scalable network VLSI implementation, Future Gener. Comput. Syst. 4 (3) (1988) 235–243.
[55] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach Learn. Res. 7 (2006) 1–30.
[56] J. Derrac, S. Garcia, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (2011) 3–18.
[57] S. Garcia, A. Fernandez, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inform. Sci. 180 (2010) 2044–2064.
[58] S. Garcia Lopez, Statistical inference in computational intelligence and data mining, http://sci2s.ugr.es/sicidm/. Last checked December 6, 2013 (online) (June 2010).
[59] H. Salimi, M. Najafzadeh, Mohsen Sharifi, Advantages, challenges and optimizations of virtual machine scheduling in cloud computing cnvironments, Int. J. Comput. Theory Eng. 4 (2) (2012) 189–193.
[60] Whats New in Performance in VMware vSphere 5.0, Technical White Paper. Available: http://www.vmware.com/files/pdf/techpaper/Whats-New-VMware-vSphere-50-Performance-Technical-Whitepaper.pdf, 2011. Last checked December 6, 2013.
[61] Google. Available: http://code.google.com/p/virtual-machine-scheduler/. Last checked December 6, 2013.

Ivanoe De Falco was born in Naples, Italy, in 1961, and got his Laurea degree *cum laude* in Electrical Engineering at University of Naples "II" in 1987. He is currently a senior researcher at the Institute of High Performance Computing and Networking (ICAR) of the National Research Council of Italy (CNR). He is the author or a coauthor of about 100 publications in international journals and in the proceedings of international conferences, and his papers have received about 220 citations in international literature. His main fields of interest include evolutionary algorithms and parallel computing. He is in the editorial board of Applied Soft Computing and of Journal of Artificial Evolution and Applications.

Umberto Scafuri was born in Baiano (AV) on May 21, 1957. He got his Laurea degree in Electrical Engineering at the University of Naples "Federico II" in 1985. He currently works as a technologist at the Institute of High Performance Computing and Networking (ICAR) of the National Research Council of Italy (CNR). His research activity is basically devoted to parallel and distributed architectures and evolutionary models.

Ernesto Tarantino was born in S. Angelo a Cupolo, Italy, in 1961. He received the Laurea degree in Electrical Engineering in 1988 from University of Naples, Italy. He is currently a researcher at National Research Council of Italy. After completing his studies, he conducted research in parallel and distributed computing. During the past decade his research interests have been in the fields of theory and application of evolutionary techniques and related areas of computational intelligence. He is author of numerous scientific papers in international conferences and journals. He has served on several program committees of conferences in the area of evolutionary computation.